Chapter 5

Authentication for People

The challenge is to devise authentication methods that exploit the strengths yet accommodate the weaknesses of our human minds and bodies. Computers are great at doing cryptographic calculations and storing keys (arbitrary bit strings) for later retrieval. Humans are not. So cryptographic protocols, which work so well for one computer to authenticate another, are impractical when a computer must authenticate a person or when a person must authenticate a computer.

Methods to authenticate an identity being claimed by a person are grouped by security cognoscente into three broad categories.

> **Something you know.** You demonstrate knowledge of a secret or fact(s) unlikely to become known to impersonators. □

> **Something you have.** You demonstrate possession of some distinctive object that is difficult for an impersonator to obtain or fabricate. □

> **Something you are.** You allow certain of your physical attributes to be measured, believing that corresponding measurements will not be similar for impersonators. □

Moreover, by using multiple authentication methods, our confidence in the result can be increased beyond what any single method provides. Users of bank ATMs are likely familiar with *two-factor authentication* involving "something you know" (a PIN) and "something you have" (a plastic bank card). Here, ignorance of the PIN makes it difficult for an attacker to benefit from stealing the bank card, and without possessing the bank card an attacker cannot benefit from learning your PIN. The general case is *multi-factor authentication*, where multiple independent[1] methods are used to authenticate a person.

[1] Recall from the Defense in Depth discussion (see page 19) that two mechanisms are considered independent to the extent that an attack to compromise one is unlikely to compromise the other.

Identity and Enrollment

We define an *identity* to be a set of *attributes*. Each attribute is a statement or property about an individual; an attribute that is associated with exactly one individual (in some presumed population) is called an *identifier*. A given person is likely to have multiple identities (e.g., as a student, as a citizen, as a credit-worthy consumer), comprising various attributes (e.g., attends classes CS513 and CS481, is a registered Democrat, always pays bills) some of which are also identifiers (e.g., Cornell University student id is 140411, social security number is 78-05-1120).

You might think in terms of having a single identity comprising all of your attributes, but virtually nobody else views you in this way. Most people will know only a few of your different identities. And by keeping these various identities separate, your friends associated with one identity need not know about your friends associated with another. Anyone who has ever tried to stay friends with both members of a couple after they have ended their relationship will know the wisdom in being able to have distinct sets of acquaintances.

Besides authentication, we might be interested in *identification*—a process by which the identity of an individual is determined from attributes that have been claimed or observed for that person. Authentication takes as input both an identity and a set of attributes (and returns whether they match), whereas identification requires only the set of attributes as input (and returns an identity). A border guard who checks whether a passport names the bearer is performing authentication; a video surveillance system attempting to locate suspected terrorists in a crowd is performing identification.

An identity is most useful if we can have confidence its constituent attributes actually hold for any individual authenticated under that identity.[2] An *enrollment protocol*, executed before the identity is added to the system, provides an initial validatation of these attributes.

- Some attributes can be checked electronically. For example, to create a new account at a web server, most of us have experienced a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart), which is a challenge-response test that is easy for a human to solve but difficult for a computer. One well known CAPTCHA displays distorted images of character sequences that are too complex for state-of-the-art image recognition software but are easily recognized by humans.

- Other attributes can only be checked by initiating physical action or provding physical evidence. For example, Alice might demonstrate that some bank account is under her control by transfering $1.00 from that account to a specified third party; Bob might validate his name by presenting a valid driver's license and passport, where each contains a photograph matching the presenter's face.

[2] For this reason, some authors prefer the term *claim* to attribute.

The design of enrollment protocols is tricky. Documents and other forms of physical evidence can be forged or altered. Moreover, documents that describe attributes of their subjects rarely offer proof that can be independently checked. A birth certificate, for example, contains no proof that the bearer is the one named in the birth certificate or that the claimed relationship between mother and child is valid.[3] Even the meaning of unaltered and valid official documents is frequently misconstrued. For example, increased confidence derived by requiring both a passport and a birth certificate is unfounded when these documents are not independent, as is frequently the case.[4]

Vendors of authentication services typically offer a choice of enrollment protocols. The choices range from protocols for obtaining low-confidence judgments about the validity of the attributes, perhaps based on credentials submitted by mail, all the way to protocols that yield high-confidence judgments based on in-person interviews, physical examinations, or even background investigations. However, it is not clear how a computer system could leverage judgments about enrollment confidence, since it is the task that usually dictates the degree of confidence sought about attribute validity. A system is more likely to benefit from having a suite of authentication protocols than having a suite of enrollment protocols, since this enables obtaining the appropriate level of confidence about the identities involved in each different task. For example, banks and other financial institutions use multi-factor authentication for customers making large withdrawals but use a single low-confidence authentication protocol for customers making deposits.

5.1 Something You Know

Knowledge-Based Authentication. With *knowledge-based authentication*, each individual, as part of the enrollment protocol, provides the system with answers to a set of queries; thereafter, to authenticate that individual, the system poses some subset of those queries and compares the responses it receives with the answers provided at enrollment. Archetypal queries concern an individual's family (e.g., "What is your mother's maiden name?"), life experiences (e.g., "In what cities have you lived?"), or current circumstance (e.g., "What is your place of employment?").

The best queries have answers that are not widely known and cannot easily become known to an attacker attempting to impersonate someone.[5] Thus, good

[3] In contrast, DNA provide an independently verifiable proof of a highly probable biological connection to a person or of a genetic relationship between two people, because new DNA samples can be collected and checked whenever confirmation of the claim is sought.

[4] A birth certificate is needed to obtain a U.S. passport, so these documents are not independent and there is little basis for increased confidence from having both.

[5] There is, for example, little benefit in asking violinist Nadja Solerno-Sonnenberg about her mother's maiden name, which would likely be "Solerno" given the hyphenated construction of "Solerno-Sonnenberg". And should the "correct" answer to a knowledge-based authentication query for a given individual become known to an attacker, it suffices to invent new queries and use them. However, such newer queries might be increasingly obscure, ultimately imposing an unacceptable memory burden on a subject, who must remember those answers.

queries become difficult to find when there is a wealth of publicly available information about individuals. And the trends here are not favorable, with the ever-expanding world wide web and with cultural norms that encourage details of individuals' lives to be revealed in blogs and through participation in online communities. With so much personal information available online, web search engines place at an attacker's fingertips the answers to lots of the queries that knowledge-based authentication systems have tended to use.

Any knowledge-based authentication scheme must have access to "correct" answers for queries it poses. This leads to a vulnerabilitioes whenever different systems happen to employ intersecting sets of queries: One system can now impersonate its users to other systems, and furthermore an attacker who compromises one system can impersonate that system's users to other systems. In addition, people worried about having personal information aggregated may object in principle to storing so much information about them on a single system, although the information presumably would be stored in some secure way.

In summary, knowledge-based authentication is convenient, since it imposes little burden on individuals to remember things, carry things, or submit their bodies to measurement. Moreover, we can increase our confidence in an authentication by requiring multiple queries to be correctly answered. But security of this scheme ultimately depends on attackers not having access to information about subjects, which is an assumption that is hard to check and might not be sound, rendering the approach unsuitable for high-confidence authentication. Nevertheless, we see it employed today, for example, in web sites where a user having forgotten his password responds to knowledge-based authentication queries to establish an identity; the web site then sends email[6] containing a password to an address that is part of that identity.

Secret-Based Authentication. An individual can be authenticated on the basis of a secret he knows, provided that secret is not known to would-be impersonators and is difficult to guess or steal. We simply check whether a person being authenticated as some identity knows the unique secret associated with that identity. Ideally, the secret is some string selected from a relatively large space of possibilities and in a manner unprejudiced by knowledge of reality or beliefs, since this makes guessing difficult.[7] Embodiments of such secrets include *personal identification numbers* (PINs), which are typically 4 digits long; *passwords*, which are likely at least twice as long and involve any characters; and *passphrases*, which are significantly longer still and also involve any characters. We use the term password in this chapter, but what is said applies to PINs, passphrases, and the other forms of secrets used for authenticating humans.

Secret-based authentication can be surprisingly subtle to implement. People have difficulty memorizing seemingly random strings. And the ever increas-

[6]This email is typically not encrypted. Thus, the password would be available to any passive wiretapper, rendering the approach unsuitable for senstive applications.

[7]The answers to knowledge-based authentication queries thus have only a superficial resemblence to such secrets, because the answers are prejudiced by knowledge of reality or beliefs and therefore the answers can be guessed by attackers.

ing power of computers means automated password guessing becomes feasible sooner than you would think. These impact how passwords are best stored, which in turn constrains the methods that can be used for checking whether a correct password has been provided.

5.1.1 Choosing a Password

One obvious way to generate a password is to choose a string randomly from some large space of possible choices. Given a set comprising N characters, there are a total of N^L length L strings of characters. For example, with the 52 lower- and uppercase alphabetic characters plus the 10 digits found on a computer keyboard, the set of strings comprising 8 characters contains $(52 + 10)^8$ or equivalently 218,340,105,584,896 (approximately 2.18×10^{14}) elements. Include punctuation symbols and blank spaces, which increasingly are being permitted in passwords, and the number of password candidates grows still larger.

A password can be generated by a computer, invented by the person it will be used to identify, or furnished by the system operator. None of these approaches is a panacea.

- People are notoriously bad at memorizing random strings, and passwords generated by a computer will seem random.[8] So computer-generated passwords are hard to memorize, hence likely to be written down. A password that is written down can be seen by others and stolen.

- Passwords invented by people are usually devised to be easy to remember—a word in the dictionary, a loved-one's name, a telephone number, a keyboard pattern (e.g., "asdf"), or some combination thereof. Passwords drawn from such a significantly smaller space are known as a *weak passwords* because they are considerably easier to guess.[9]

- A password furnished by a system operator must be either computer-generated or selected by some person; one of previous two cases then applies. Moreover, an operator-selected password is best changed immediately by the recipient, to prevent the operator from subsequently abusing knowledge of that password. One of the above cases would apply to that new password, too.

[8] Some have suggested that the computer employ rules to produce passwords that are easy to remember because they are pronounceable. For example, every consonant might be followed by a vowel and two adjacent letters might never both be vowels or consonants. Or, the computer might generate passwords that are random sequences of existing words. However, imposing such restrictions is counterproductive, since the amount of work to enumerate the space of possible password choices is now reduced, making that attacker's job easier.

[9] The 20 volume *Oxford English Dictionary* (second edition) contains on the order of 6.15×10^5 words. Compare that with the 146,813,779,479,510 (approximately 1.46×10^{14}) strings having 10 or fewer alphabetic characters, and it becomes clear that there are considerably more random alphabetic strings than dictionary words.

For reasons of convenince, most systems allow people to invent their own passwords. Additional means must then be used to defend against attackers guessing those weak passwords.

In *on-line guessing* attacks, the system itself is used by the attacker to check password guesses. One defense here is to limit the rate or number of password guesses allowed, thereby reducing the chances that an attacker will make a correct guess.

- Make authentication a time-consuming process by requiring that passwords be entered manually, that checking a password require an intrinically long computation, and/or that the system's password checking routine include a `sleep` operation or some other delay.

- Impose a limit on unsuccessful guesses and, when that limit is reached, disconnect from the source and refuse later attempts to authenticate that identity (until a system operator intercedes).

The second measure also increases the chances an attack will be detected while it is in progress or soon thereafter. Note, however, that limiting the number of unsuccessful password guesses for a given identity creates an opportunity for denial of service attacks; an attacker whose incorrect password guesses exceed the limit blocks all authentications for that identity, which prevents the *bona fide* owner of that identity from accessing the system.

By restricting what information is returned when an authentication attempt fails, we further impede on-line guessing. Some systems prompt for the identifier being authenticated and reply with an error indicating if it is not known to the system. Such replies tell attackers what identifiers could serve as targets of attack. A better design is for the system to request the identifier and corresponding password together; only after receiving both, does the system reply. And if either the identifier is unknown or the password is erroneous then a single, minimally-informative response is returned (e.g., "Authentication attempt failed.").

Covert channels provide another way an attacker might determine an identifier is valid. For example, if a system does not check the password when the identifier is invalid, then a longer elapsed time for processing an authentication request indicates when it is worth trying other password guesses for that same identifier. This vulnerability is eliminated by adding execution delays to create either uniform or random response times. Another example of a covert channel appeared in the `TENEX` operating system, where a password was a required argument to the `connect` system call. The implementation compared the password argument, character by character, with the correct password; it terminated when the first erroneous password character was found. If the password argument spanned a page boundary then a page fault occurred whenever the argument's characters appearing on the first page were a prefix of the password. Because the occurrence of a page fault could be sensed by the attacker and the attacker could control the alignment of a password argument on a page, a covert

channel enabled an N character password to be deduced in $26N$ guesses instead of the presumed 26^N guesses.[10]

A system's response to a successful authentication attempt should give (i) the times when the last successful authentication for this identifier occurred, and (ii) the times when authentication was unsuccessfully attempted since. In reporting the last successful authentication, a user (who presumably will recall when he last authenticated that identifier) can detect a prior successful on-line guessing attack; in reporting past unsuccessful authentications, the user becomes aware that on-line guessing attacks had been attempted.

The strongest defense against attackers guessing weak passwords is to enforce a prohibition against users choosing them. To implement this, rules such as those in Figure 5.1 can be employed to prevent users from creating easily guessed passwords. Some of the rules are obvious (e.g., avoid passwords that appear in a dictionary); other rules correspond to memory tricks that people use and attackers know people use (e.g., replacing a letter by a number that looks similar, such as replacing "e" in by "3" or "l" by "1"). No set of syntactic rules, however, will completely characterize easy-to-guess passwords, and user laziness about memorization provides a strong incentive for discovering classes of passwords that are easy to remember but the rules do not reject. Sooner or later, attackers too discover these password classes and include such passwords in their on-line guessing attacks.

5.1.2 Storing Passwords

The obvious scheme for storing passwords is to use a file that contains the set of pairs $\langle uid_i, pass_i \rangle$, where uid_i is an identifier and $pass_i$ is the associated password. The file system's authorization mechanism is then used to restrict which principals can access this *password file*, as follows.

- An ordinary user must not be able to read the password file or its backups. This prevents that user from obtaining the password for any identifier.

- An ordinary user must not be able to write the password file or its backups. This prevents that user from changing the password for an identifier (to then know that password).

- The program that authenticates users must be able to read the password file.

- The program used to add/remove user identifiers and change passwords must be able to read and write the password file.

[10] The first character of the password argument is placed as the last character of a page, and `connect` is invoked. In at most 26 tries, the correct first character will be identified because a page fault occurs. The password argument is next aligned so the first two characters appear as the last characters of a page; another 26 guesses suffice to find that prefix. After N steps, each requiring at most 26 guesses, the length N password is known.

1. Strings easily derived from words or sequences of words appearing in an English or foreign language dictionary, because they appear verbatim or reversed, perhaps with the following transformations applied:
   (a) deleting vowels or spaces,
   (b) capitalizing some letters,
   (c) adding a suffix and/or prefix,
   (d) replacing one or more letters by similar looking non-alphabetic characters (e.g., "0" for "o") and/or by homophonic letters or clauses (e.g., "4" for "for").
2. Strings derived (in the sense of 1) from the user's identity—name, initials, login id, or other attributes.
3. Strings derived (in the sense of 1) from acronyms, names of people, or names of places.
4. Strings shorter than a certain length (e.g., 8 characters).
5. Strings that do not contain a mix of numbers, special symbols, upper- and lowercase characters.
6. Strings produced by typing simple patterns on the keyboard.
7. Strings comprising only numbers.
8. Strings that mix numbers and characters but resemble license plates, office numbers, etc.

Figure 5.1: Rules Characterizing Weak Passwords

Security here requires trusting that the file system authorization mechanism has no vulnerabilities, that the access-authorization policy for the password file is correctly set whenever that file is edited and stored, that programs authorized to read the file do not leak its contents, and that traces of file contents are not left to be read from unallocated memory or unallocated disk pages that previously stored the password file. Some are troubled by the need for all this trust, but if the trust is not misplaced then storing passwords in this way leaves only one avenue to attackers: on-line guessing.

An alternative to using the file system's authorization mechanism for protecting the confidentiality of passwords is to compute a cryptographic hash function $\mathcal{H}(pass_i)$ for each each password $pass_i$ and store the set of pairs $\langle uid_i, \mathcal{H}(pass_i)\rangle$ as the password file. Because by assumption $\mathcal{H}(\cdot)$ is a one-way function, knowledge of $\mathcal{H}(pass_i)$ will not reveal enough about $pass_i$ to be useful attackers.

Letting *HashPwd* denote the set of $\langle uid_i, \mathcal{H}(pass_i)\rangle$ pairs stored in the password file, a person P is deemed to be authenticated if and only if P en-

1. $P \longrightarrow Sys$: uid, $pass$
2. Sys: **if** $\langle uid, \mathcal{H}(pass)\rangle \in HashPwd$ **then** P deemed authenticated.

Figure 5.2: Hashed Password Authentication Protocol

ters some identifier uid and password $pass$, and the system determines that $\langle uid, \mathcal{H}(pass)\rangle \in HashPwd$ holds. This depicted in Figure 5.2.

Historically, MD5 or SHA-1 were used for $\mathcal{H}(\cdot)$, but cryptoanalyists established in 2005 that MD5 or SHA-1 do not satisfy the collision resistance properties required of cryptographic hash function. Some other cryptographic hash function should therefore be used. An alternative, albeit slower, is to use a shared key encryption algorithm configured to encrypt some constant C, using $pass$ (or something derived from it) as the key; early versions of UNIX implemented $\mathcal{H}(\cdot)$ by iterating DES several times with 0 for C. Recall, though, that faster is not necessarily better when it comes to validating a password—on-line guessing attacks are hindered by slower password checking, because then fewer guesses can be attempted in a given interval.

Salt. Unrestricted read access to hased password file *HashPwd* still enables *off-line guessing* attacks, also known as *dictionary* attacks. The attacker first obtains a list[11] of candidate passwords or, using rules like those in Figure 5.1, constructs such a list; this list is then used by the attacker in computing a *dictionary*, which is a set *Dict* of pairs $\langle w, \mathcal{H}(w)\rangle$ for each word w in the list of candidate passwords. Finally, the attacker reads *HashPwd* from the system to be attacked and generates the set of pairs

$$\{\langle uid, pass\rangle \mid \langle uid, \mathcal{H}(pass)\rangle \in HashPwd \ \wedge \ \langle pass, \mathcal{H}(pass)\rangle \in Dict\} \qquad (5.1)$$

by identifying elements having the same $\mathcal{H}(pass)$ value in both *HashPwd* and *Dict*. Each $\langle uid, pass\rangle$ pair in (5.1) enables the attacker to impersonate identifier uid.

The computation of *Dict* and (5.1) constitute an up-front cost to the attacker. However, such an investment often can be justified.

- The work to compute *Dict* is amortized, because the single *Dict* suffices for attacking every system whose password file is computed using cryptographic hash function $\mathcal{H}(\cdot)$. All instances of a given operating system typically use the same hash function, and therefore a single *Dict* can be used in attacking any target running that operating system.

- Beyond obtaining a copy of *HashPwd*, the computation involved in actually attacking a given system—the construction of (5.1)—need not be performed on that target system. The chance that an attacker's preparations will be discovered, hence the risk to the attacker, is thus reduced.

[11] In 2007, lists found on attacker web sites contained as many as 40 million entries.

1. $P \longrightarrow Sys$: uid, $pass$
2. Sys: $\langle uid, n, p\rangle := HashSaltPwd[uid]$
3. Sys: **if** $\mathcal{H}(pass \cdot n) = p$ **then** P deemed authenticated.

Figure 5.3: Hashed Password Authentication Protocol with Salt

Moreover, the attacker is now not limited by a target system's computational power or deliberately slow implementation of $\mathcal{H}(\cdot)$.

- The attack is likely to yield multiple user identifiers with passwords for the target system, so it provides the attacker with multiple avenues of compromise. This is ideal for the attacker who does not need to impersonate a specific user, and many attacks require only the initial access that impersonating any user provides. So the offline guessing attack succeeds if any user identifier has a weak password.

One defense against off-line guessing attacks is to change the password file in a way that makes the computation of (5.1) infeasible. To accomplish this, we might store with each uid_i a nonce n_i, called *salt*, and combine that nonce with $pass_i$ before computing cryptographic hash function $\mathcal{H}(\cdot)$. The password file now stores a set *HashSaltPwd* of triples, $\langle uid_i, n_i, \mathcal{H}(pass_i \cdot n_i)\rangle$. Early versions of Unix used 12-bit numbers for salt; the nonce for a given *uid* was obtained by reading the real-time system clock when creating the account for *uid*.

We extend the protocol of Figure 5.2 as shown in Figure 5.3 to accommodate the per *uid* salt, where $HashSaltPwd[uid]$ denotes the unique triple having the form $\langle uid, \ldots\rangle$ in *HashSaltPwd*.

With b bit nonces for salt, off-line guessing requires the attacker to compute in place of *Dict* the considerably larger set *SaltDict* of triples $\langle w, n, \mathcal{H}(w \cdot n)\rangle$ for each candidate password w and each value n satisfying $0 \leq n \leq 2^b - 1$. So *SaltDict* is 2^b times as large as *Dict* and thus considerably more expensive to compute and store. Analogous to (5.1), the attacker now constructs the set of pairs

$$\{\langle uid, pass\rangle \mid \begin{array}{l} \langle uid, n, \mathcal{H}(pass \cdot n)\rangle \in HashSaltPwd \\ \wedge\ \langle pass, n, \mathcal{H}(pass \cdot n)\rangle \in SaltDict\} \end{array} \tag{5.2}$$

and element $\langle uid, pass\rangle$ in that set enables the attacker to impersonate corresponding user *uid*.

Pepper. The cleartext salt in *HashSaltPwd* does not defend against what we will call a *limited off-line guessing* attack, where one particular system is the target. The attacker here needs to compute only a subset of *SaltDict*—the subset containing only those triples $\langle w, n, \mathcal{H}(w \cdot n)\rangle$ for which salt n appears in *HashSaltPwd* on the target system. If *HashSaltPwd* contains N elements then this subset of *SaltDict* is only N times the size of *Dict*. For b bits for salt, $N \ll 2^b$ is likely to hold, so the attacker replaces an expensive computation

1. $P \longrightarrow Sys$: uid, $pass$
2. Sys: **if** $(\exists\, n :\ \langle uid, \mathcal{H}(pass \cdot n)\rangle \in HashPepperPwd)$
   then P deemed authenticated.

Figure 5.4: Hashed Password Authentication Protocol with Pepper

with one that is considerably cheaper. Specifically, the attacker incurs a cost proportional to $N|Dict|$ for computing a *SaltDict* that works against the one target system of interest, instead of incurring a cost proportional to $2^b|Dict|$ for the *SaltDict* that would work against all targets.

To defend against limited off-line guessing attacks, we might keep the salt secret by storing a set *HashPepperPwd* of pairs $\langle uid_i, \mathcal{H}(pass_i \cdot n_i)\rangle$, where nonce n_i, now called the *pepper*, is not stored elsewhere in the tuple for uid_i (as salt would be). A protocol for authenticating *uid* given a password *pass* is given in Figure 5.4; step 2 deems P authenticated if and only if there is some pepper n for which $\langle uid, \mathcal{H}(pass \cdot n)\rangle$ is an element of *HashPepperPwd*. To determine whether such an n exists, the system must enumerate and check possible values. This search is best started each time at a different, randomly selected place in some standard enumeration of possible pepper values; otherwise, the delay to authenticate *uid* conveys information about whether n is early or late in that standard enumeration of possible pepper values, thereby reducing the size of the search space, hence the work required of an attacker.

When pepper is used, the amount of work required to build a dictionary *PepperDict* for an off-line guessing attack is proportional to the number of pairs $\langle w, \mathcal{H}(w \cdot n)\rangle$, for w ranging over the candidate passwords and n ranging over all possible pepper values. For b bits of pepper, this means that *PepperDict* is a factor of 2^b larger than *Dict*; the attacker's work has been increased exponentially. Moreover, unlike salt, pepper defends against limited off-line guessing attacks, because the absence of cleartext pepper in *HashPepperPwd* means an attacker has no way to generate the target-specific smaller dictionary that enables a limited off-line guessing attack. So b bits of pepper constitutes a better defense than b bits of salt.

Pepper is not a panacea, though. The number of possible pepper values affects the time required to perform step 2 of the authentication protocol in Figure 5.4. If this delay is too large, then users will lose patience. So the number of bits of pepper must be kept small enough for step 2 to complete within a few seconds, and the potency of pepper as a defense is limited. Compare this to salt. The number of possible salt values is virtually unbounded, being constrained only by the storage required for *HashSaltPwd*.

Putting it Together. By combining salt and pepper, we obtain a better defense than results when either is used alone. Salt $salt_i$ and pepper ppr_i are both inputs to the cryptographic hash computed for each password $pass_i$; passwords are stored in a set *HashSsndPwd* of triples $\langle uid_i, salt_i, \mathcal{H}(pass_i \cdot salt_i \cdot ppr_i)\rangle$; and the protocol of Figure 5.5 is used for authentication.

1. $P \longrightarrow Sys$: uid, $pass$
2. Sys: $\langle uid, s\rangle := HashSpicedPwd[uid]$
3. Sys: **if** $(\exists\, ppr :\ \langle uid, s, \mathcal{H}(pass \cdot s \cdot ppr)\rangle \in HashPepperPwd)$
   then P deemed authenticated.

Figure 5.5: Hashed Password Authentication Protocol with Salt and Pepper

If there are bs bits of salt, bp bits of pepper, and N tuples in $HashSsndPwd$, then a dictionary $SsndDict$ for an off-line guessing attack would be 2^{bs+bp} times larger than $|Dict|$, and the subset of $SsndDict$ required for a limited off-line guessing attack would be $2^{bp}N$ times larger than $|Dict|$. Off-line attacks are made infeasible by choosing bs large enough. Defense against limited off-line guessing attacks, however, depends on the number (2^{bp}) of possible pepper values which, unfortunately, is limited by the processing speed of the target machine and the delay that users are willing to tolerate for step 3 of Figure 5.5. Even stronger security requires some form of defense in depth, such as using file system authorization to restrict read access to a password file that stores cleartext salt or any form (i.e., hashed, salted and/or peppered) of password.[12]

5.1.3 Authenticating Requests for Passwords

A program that requests your password might be running on behalf of an operating system that is trying to authenticate you. Or, it might be a *password harvester* masquerading as the operating system and running on behalf of an attacker who is attempting to steal your password. Human users thus need some means to authenticate the source of password requests, and that is the subject of this section. We shall see that because humans and computers have such different capabilities, the means by which humans can authenticate computers are very different from means computers can employ to authenticate humans.

Trusted Path. A *trusted path* is a trustworthy communications channel between an I/O device, like a keyboard and display, and a known program, like an operating system's authentication routine. One way users can avoid being spoofed by password requests from an attacker's program is to open[13] such a trusted path and communicate using it. The components and protocol steps are given in Figure 5.6.

The security of this scheme depends on trusting the keyboard driver and OS authentication routine. An attacker who alters either can build a password harvester that, to a human user, appears to behave like what is outlined in

[12] For example, the shadow password file `/etc/shadow` in LINUX and `/etc/master.passwd` in BSD Unix systems are read protected, and they store cleartext salt along with hashed, salted passwords. Both systems continue to support `/etc/passwd`, which is generally readable but does not contain any form of the password for each user id.

[13] In Microsoft's Windows NT operating system and its successors, for example, typing `Ctrl-Alt-Delete` serves this purpose.

1. *User*: Activate the *secure attention key* (`SAK`) at the keyboard.
2. *OS keyboard driver*: Intercept each `SAK` activation and start the OS authentication routine. Display a password request prompt (perhaps in a separate window). Then monitor the keyboard for the user's response.
3. *User*: Enter a user id and corresponding password.
4. *OS keyboard driver*: Forward that user id and password to the OS authentication routine.
5. *OS authentication routine*: Validate the user id and password.

Figure 5.6: Trusted Path Component Authentication Protocol

Figure 5.6. You might think that rebooting the computer from its disk would defend against such a compromised keyboard driver or OS authentication routine. However, rebooting from disk does not defend against an attacker who has corrupted the version of the operating system stored on the disk. Nor does rebooting from some other (known to be clean) I/O device suffice, if the attacker has modified the boot loader to read the corrupted boot image on disk no matter what boot device is specified. Defend the boot loader and an attacker can force the wrong boot loader to execute by exploiting `BIOS` vulnerabilities; defend the `BIOS`, and an attacker's ability to modify the hardware remains problematic.

What then is the benefit of providing support for a trusted path? First, physically limiting access to the hardware does prevent an attacker from modifying it, so one class of attacks can be ruled out. Second, if we trust the hardware, thus assume it cannot be compromised, then we can defend against the other attacks outlined above by leveraging a secure co-processor to detect modification or replacement of system software.[14] Finally, repositioning a vulnerability from a higher layer (e.g., the operating system) down to a lower layer (e.g., the `BIOS`) typically increases the security of the system, because exploiting vulnerabilities at lower system layers usually requires more sophistication hence exceeds the expertise of some threats.

Visual Secrets. Visual appearance is another way that users can authenticate the source of password requests. Here, the manner in which the request is rendered serves as a *visual secret* that is shared between requester and responder. The secret might be a formatting convention, some specific text, and/or an image; or the secret might embody how to access routines that provide the only way for controlling some discernable aspect of certain outputs.

An executing program demonstrates knowledge of the visual secret by how or where the password request is rendered. The human responder validates the password request by inspecting it and ignoring those password requests whose

[14] This is discussed in §**??**.

appearance departs from what is expected.

An attacker's password request can now succeed only by demonstrating knowledge of the appropriate visual secret. That can be made difficult in two ways.

- *Share a different visual secret with each user.* Here, the protocol for a password request involves two separate steps: (i) request a user id, and (ii) request the password for that user id by displaying the appropriate shared visual secret for the given user id.[15] An attacker succeeds only by obtaining the visual secret for each user id to be spoofed, something presumed to be just as difficult as obtaining the corresponding password by means other than a spoofed request.

- *Prevent arbitrary programs from rendering certain text or images in certain regions of a display.* By including in a password request some content that can be rendered only by a program demonstrating knowledge of a secret, we prevent an attacker's program from generating a legitimate password request.

These schemes work when humans can and do distinguish among various possible renderings of password requests. Yet humans have a tendency to overlook minor details when reading text or viewing images. That enables attacks. For instance, most people will not differentiate between characters output by a program and an embedded image containing those same characters. Thus, by embedding images instead of outputting characters, a program might be able to circumvent defenses where knowledge of the secret is being demonstrated by printing a character sequence. Experience with web browsers shows that people also don't look at status fields—much less look at them carefully—to see whether they display expected values. A tamper-proof status field for the name of a requester is, therefore, unlikely to defend against bogus requesters whose names are spelled similarly to *bona fide* requesters.

Our human insensitivity to slight differences in rendering helps enable *phishing* attacks, where attackers attempt to acquire private information by directing users to a web site designed to resemble the web site for some legitimate institution. Users are induced to visit the attacker's web site by being presented with a link that closely resembles a link for the site being spoofed[16] (e.g., `www.paypa1.com` versus `www.paypal.com`), they are not careful about checking this, and they enter private information (a password, perhaps) which then becomes known to the attacker.

Humans are quite good at recognizing and distinguishing pictures of different scenes or people. Thus, certain per-user visual secrets—those involving larger, richer images—seem promising as a means for a human to authenticate request sources. Moreover, improvements in user interfaces in conjunction with

[15] To prevent attackers from determining whether a particular user id is valid, the system must not behave differently in step (ii) when a unknown user id is entered in step (i). One solution is for the system to create a new visual secret for that invalid user id.

[16] This is sometimes known as a *cousin-name* attack.

an expanding population of more-experienced users ought to mean that, in the future, users will be better at detecting differences in how requests are rendered.

5.1.4 Password Pragmatics

A single password is far easier to devise and remember than multiple passwords. So it is tempting to stick with one password, reusing it for all your various different identities (e.g., computer accounts on different machines, various web sites, etc.). However, an attacker now gains great leverage from compromising that single password. There are two obvious defenses:

- If each password has only a limited lifetime, then a compromised password sooner or later will become useless to an attacker—even if that compromise never becomes known.
- If separate passwords are employed for different identities, then an attacker who compromises one password could impersonate only the one identity.

How might systems support these practices?

Password Expiration. Some systems periodically force users to select new passwords, either after a fixed interval has elapsed (typically, months) or after the password has been used some specified number of times. By storing a user's expired passwords, the system can prevent new passwords from resembling their predecessors.[17] Note the trade-off between security and convenience. Passwords that must be changed frequently are better for security, but frequent memorization of new passwords is inconvenient. Moreover, if users finds all this memorization too burdensome and write-down their current password then they undermine security by facilitating password theft.

Infrequent users pose a special problem for password expiration schemes. The infrequent user might well resent having to access the system periodically solely for the purpose of replacing a password before it expires. Yet to allow authentication using an expired password—even if the authenticated user must immediately select a new password—is risky, because an attacker who has compromised an old password would now get to pick the new password, thereby circumventing the defense password expiration was intended to provide.

Password Hashes. An individual can, with a minimum of effort, create a distinct password $pass_I$ for each identity I by (i) memorizing a single secret s and (ii) using a function $\mathcal{F}(\cdot)$ to combine s and I and derive password

[17] However, this functionality can be circumvented by resourceful users. For example, if passwords expire every month, then passwords `Mycroft01`, `Mycroft02`, ..., `Mycroft12`, are easy for a user to generate once `Mycroft` has been memorized, and a system might deem each of these different enough from its predecessor. Unfortunately, passwords forming a sequence that is easy for a user to generate are probably also easy for an attacker to reconstruct from a single compromised password in the sequence. An attacker who successfully compromises `Mycroft02` in February, for instance, would be wise to try `Mycroft05` in May.

$pass_I = \mathcal{F}(s, I)$. You might, for example, select `Moriarty` as your secret, memorize it, and then (as needed) produce password `MwowrwieaBratyycom` for your identity at `www.eBay.com`, `MwowrwiCaNrNtcyom` for `www.CNN.com`, and so on.[18] However, as noted above, such schemes risk producing weak passwords if a single compromised password provides enough information about both secret s and combining function $\mathcal{F}(\cdot)$ so that an attacker can deduce passwords associated with other identities.

Were $\mathcal{F}(\cdot)$ one-way, then even if $\mathcal{F}(\cdot)$ were publicly known, an attacker who compromises one or more passwords would learn nothing useful about the secret input s to $\mathcal{F}(\cdot)$ used to produced those passwords. The attacker would thus be no better off for deducing passwords for other identities. Cryptographic hash functions are one-way. Thus, given a cryptographic hash function $\mathcal{H}(\cdot)$, a strong password for each of a principal's identities can be obtained if the principal memorizes a single secret s and uses *password hash* $\mathcal{H}(s \cdot I)$ as the password for identity I. In particular, the preimage resistance of $\mathcal{H}(\cdot)$ would make it infeasible for an attacker to calculate $s \cdot I$, hence learn s, from knowledge of password $\mathcal{H}(s \cdot I)$.

Few humans will be able to calculate $\mathcal{H}(s \cdot I)$ mentally, nor would very many be willing to undertake a paper and pencil calculation in order to be authenticated each time. This restricts the applicability of password hashes to settings where a computer is available to aid in generating passwords: The human principal inputs the secret s along with the identity I for which a password is sought; the computer provides the sought password by evaluating $\mathcal{H}(s \cdot I)$ and then erasing s and $\mathcal{H}(s \cdot I)$ from its memory (so that later attacks on the computer can reveal neither).

Password hashes are the basis of `PwdHash`, a web browser plug-in to enable users who memorize only a single secret to authenticate themselves using a different password at each different web site. Password fields in web page forms are usually delimited by a special HTML tag. This tag allows the `PwdHash` plug-in to replace any text *pass* a user enters into such a password field with $\mathcal{H}(pass \cdot url)$, where *url* is the url of the web page being visited. The information sent back to the web site thus contains a site-specific password $\mathcal{H}(pass \cdot url)$, even though the human user has not memorized or entered any site-specific passwords. Moreover, such automatic generation of site-specific passwords defends against certain kinds of phishing attacks. This is because the phishing web site and the site being impersonated will have different url's. Therefore, the password sent to the phishing web site will be different from the password that would have been sent to the site being impersonated—phishing sites thus cannot learn passwords that can be used to impersonate a user on other sites.

Password Hygiene. To focus only on system mechanisms for preventing password compromise ignores a significant vulnerability: human users who might reveal their passwords through action or inaction. The defense against such attacks is twofold: user education about easily exploited behaviors and the

[18] Here, function $\mathcal{F}(s, I)$ interleaves its arguments s and I.

adoption of practices to discourage that behavior. Both are covered in what follows.

Sharing Passwords. The human penchant for being helpful can sometimes become a vulnerability. A common attacker's ploy is to telephone a user and pretend to be a system operator needing that user's password in order to fix an urgent problem. The user, who like most people wants to be helpful, reveals his password. In a variant, the attacker calls a system operator, pretending to be a *bona fide* user with an urgent need to access the system but has forgotten his password. The operator, trying to be helpful, resets that user's password and informs the attacker of the new password. Neither of these attacks could succeed if system designs never required one user to reveal his password to another and, therefore, users could know never to reveal their passwords to anyone.

Some users who share their passwords with others are simply confused about the difference between authorization and authentication. We all have occasion to leave our house keys with people we trust, so they can take in the mail or feed Mena the cat. The lock on the front door is an authorization mechanism; it grants access to those who possess our house key. So a password, because it unlocks access to a computer, is equated (erroneously) with a house key; sharing the password is then presumed (erroneously) to be a way others can be granted computer access. In fact, a computer's login authorization mechanism grants computer access to users having certain identities, and sharing your password is tantamount to granting your identity to somebody else. The real problem is user ignorance.

Recording Passwords. Few people are able to memorize even a modest number of strong passwords. The rest of us choose between two options:

1. reuse a few strong passwords (because memorizing those is a manageable prospect) rather than having a separate password for each identity, or

2. record passwords in some written or electronic form, so there is no need to memorize them.

Each option brings vulnerabilities that enable a password-based authentication to be compromised.

Option 1 sacrifices a form of defense in depth. When the same password is used for multiple identities, then after compromising the password for one identity, an attacker can be authenticated under the other identities as well. If different identities have access to different resources, then this undermines the Principle of Least Privilege. Yet there are contexts where having the same password for multiple identities is perfectly reasonable—notably, situations where having multiple, distinct identities was not the user's choice. For example, many newspapers and other web-based information providers create a distinct identity for every user they serve, and these sites typically employ password authentication; no harm is done if a user who would have been satisfied with having a

single identity to access some set of sites chooses the same password for all of them.

With option 2, our concern should be defending against attackers accessing the recorded password list. One classic example is the handwritten note so often found affixed to a system console in a computer room, giving passwords for system operator accounts. Anyone who gains entry to that computer room (say, by convincing the janitor to unlock the door) can use these passwords to become a system operator. Another example is the printed list of passwords that is carefully locked away until it is discarded (without first being shredded), so the list becomes available to any attacker willing to rummage through the trash.[19] Defense in depth suggests that both encryption and physical means (i.e., physical lock and key) be used to protect recorded password lists.

Observing Passwords. Even a password that has not been written down is potentially vulnerable to theft by attackers who can observe or monitor emanations from a system. Some of these attacks are straightforward; others require considerable expertise in signal-processing.

- An attacker might peer over the user's shoulder and watch the password being typed or might use a telescope to watch from a distance. This is known as *shoulder surfing*. The obvious defense is to shield from view the keyboard being used to enter a password. Keyboards on ATMs, for example, are often positioned so they cannot be seen by others when somebody is using the ATM.

- Instead of watching the keyboard, an attacker might instead observe the system's display as a password is typed. Direct visual observation is not required. Researchers have been able to recover the contents of a CRT display by monitoring the diffuse reflections from walls, and they have been able to recover the contents of an LCD display by monitoring RF radiation from the video cable connecting the LCD to the computer. Here, the obvious defense is not to echo the typed characters on the display when a password is entered.

- Each key on a keyboard will have a slightly different sound when pressed, and programs exist to perform the signal processing needed recover a user's keyboard inputs from recordings of such acoustic emanations. Background noise (or music) is easily filtered out, so it is not an effective defense. Physical security to prevent an adversary from planting a microphone or other listening device seems to be the best defense against such attacks.

[19] The practice of combing through commercial or residential trash in search of confidential information is referred to as *dumpster diving*.

5.2 Something You Have

An artifact can be used for authenticating someone if it cannot be counterfeited by threats of concern. Give an *authentication token* $\mathcal{A}_P$ to a person having identity P (but to nobody else) and, thereafter, anyone able to prove possession of $\mathcal{A}_P$ can be authenticated as having identity P. Something you have speaks for you and, therefore, can be used to authenticate you.

This does mean that an attacker who steals $\mathcal{A}_P$ is able to impersonate P. And arguably, it is easier to steal an artifact than to steal a secret (which is materialized only when it is being input). Theft of a secret, however, might not be noticed, whereas P is likely to notice if $\mathcal{A}_P$ is missing; and once the theft is noticed, P can foil would-be impersonators by informing a system administrator that $\mathcal{A}_P$ no longer speaks for P.

We defend against attackers using stolen authentication tokens by employing multi-factor authentication. Typically, knowledge of a PIN and possession of the authentication token are both required, yielding 2-factor authentication. In some systems, users choose two secrets—a normal PIN and a *duress* PIN. Entering the duress PIN instead of the normal PIN causes authentication to proceed but (i) alerts system operators that an attacker is forcing the user to authenticate, and (ii) authenticates the user to a version of the system that gives the appearance of operating on real data but really isn't, rendering ineffectual attacker-coerced operations that follow.

Authentication tokens (also known as *security tokens*, *hardware tokens*, or *cryptographic tokens*) come in various shapes and sizes, including printed pages of values, plastic credit cards with magnetic stripes, radio-frequency identification (RFID) chips, fobs containing small LCD displays, dongles with USB connectors, and calculator-like devices having keypads and displays. What all of these have in common is the ability to store a value; this value is associated with the identity the token authenticates. How this value is sensed by a computer doing authentication depends on the token's physical form. Sometimes special hardware is required: authentication tokens having magnetic stripes must be swiped through a special reader, and RFID chips are interrogated using a radio transceiver tuned to the appropriate frequency. But for other types of authentication tokens, special hardware is not needed. Authentication tokens having LCD readouts, for example, can display a value which a user reads and then conveys to the authenticating computer using that computer's keyboard; dongles that plug into USB connectors become I/O devices the computer's operating system reads directly.

The value an authentication token stores might well exceed what a human can reasonably be expected to memorize. The authentication token, in effect, augments the human's memory. Some types of authentication tokens go even further in compensating for human limitations. An authentication token might perform cryptographic operations, thereby enabling cryptography to be used for authenticating the token holder, and it might read and store information from the computer to which it connects, so the the problem of a computer authenticating a person is now transformed into the problem of one computer

authenticating another.

Authentication tokens employing digital circuits require electric power. Batteries are an obvious power source, but they are relatively large and, because they have finite lifetimes, must be replaced or recharged periodically. Alternatively, the channel that connects an authentication token to the computer authenticating it could provide power. Tokens with USB connectors can get power from the USB; so-called *passive* RFID tokens get their power from the radio signal that interrogates them.[20] Such intermittent power sources, however, cannot be used for digital circuits that require power to store values between authentications.

Magnetic stripes can be used to store information, and they make no power demands on the authentication token.[21] But a disadvantage of magnetic stripes is their low storage density. The magnetic stripe on a typical credit card, for example, is able to store only a few hundred characters. Moreover, magnetic stripes cannot be used to perform calculations.

The design of an authentication token invariably involves trade-offs. Understanding interactions among the various dimensions of the design space is thus instructive, whether you are selecting among existing authentication tokens or designing from scratch.

- *Form Factor.* Users are primarily interested in convenience. Smaller and lighter tokens are preferred. The way information is transferred between the token and the computer also influences convenience: radio is the most convenient, physical connections less so, and least convenient would be when users must manually transfer information by typing into the keyboard on one device what they see on another device's display. Finally, periodic maintenance operations, for example to load new secrets or deal with dead batteries, are seen as a burden.

- *Computational Capabilities.* System designers are concerned with what computations the token can perform and how it communicates. Stronger authentication methods require cryptographic operations. Also of interest is the channel between the authentication token and the authenticating computer: Is that channel secure against passive wiretappers? What about active wiretappers?

- *Economics.* Cost differences become significant when large numbers of authentication tokens need to be procured. However, less expensive tokens—e.g., RFID chips and cards having magnetic stripes—require more expensive, specialized hardware at the authenticating computers, so the ratio of authentication tokens to authenticating computers becomes relevant. Less expensive tokens typically are easier to counterfeit, creating a tension between cost and security.

[20] Other *active* RFID tokens have batteries.

[21] With magnetic stripe memory, power is required by the I/O device used to read and update the magnetic stripe. This I/O device is typically connected to and powered by the authenticating computer.

Finally, authentication tokens can be used for authenticating arbitrary objects, but the design trade-offs are typically quite different. For instance, an authentication token can be incorporated into an object, hence difficult for an attacker to detach and deploy elsewhere, so multi-factor authentication (with its need for additional input mechanisms) would not be necessary. RFID chips seem the preferred technology for authenticating objects because of the low per-object cost and because communication with an RFID chip does not require a physical connection. Example applications include attaching RFID tags to items in warehouses and stores for inventory management, to trans-modal shipping containers for customs clearance and homeland security, and to airline luggage tags for bag routing and recovering lost bags.

5.2.1 Tokens that Generate One-Time Passwords

Many authentication tokens are simply devices for generating a sequence of *one-time passwords*.

Properties of One-Time Passwords.

- A one-time password may be deemed valid only once—the first time it is used.
- Attackers cannot predict future one-time passwords, even with complete knowledge of what passwords have already been used. □

Periodically changing an (ordinary) password bounds the interval during which an attacker who learns that password could be authenticated as you, and a one-time password can be seen as the limit-case of this defense. Replay attacks, for example, are impossible with one-time passwords, since replays reuse passwords. One-time passwords are thus attractive when the channel from the principal being authenticated could be monitored by an adversary. Man-in-the-middle attacks do remain possible, however.

One-time passwords can be generated on-the-fly as needed or they can be precomputed and stored for later use. In either case, two operations must be implemented: an operation for providing fresh one-time passwords to users (but not attackers), and an operation for checking the validity of a one-time password when it is used. Humans, with our poor memories and limited calculational abilities, are not likely to be good at either operation. Authentication tokens could be. So one-time passwords can be used to authenticate a human P by virtue of something that P has—an authentication token $\mathcal{A}_P$. We now explore various instances of this general approach.

One-Time Passwords from Challenge-Response

The underlying basic challenge-response protocol for one computer A authenticating another B (see §**??**) can be seen as implementing a one-time password scheme, where the response B generates in reply to A's challenge is the next valid one-time password for B. Moreover, an authentication token $\mathcal{A}_P$ endowed

1. $Sys \longrightarrow \mathcal{A}_P$: r where r is unpredictable
2. $\mathcal{A}_P \longrightarrow P$: "Enter PIN on keypad"
3. $P \longrightarrow \mathcal{A}_P$: PIN_P
4. $\mathcal{A}_P$: **if** $\mathcal{H}(PIN_P) \neq hpin_P$
   then $\mathcal{A}_P \longrightarrow P$: "Entered invalid PIN"
   else $\mathcal{A}_P \longrightarrow Sys$: $\langle P, resp_r \rangle$ where $resp_r = \mathcal{H}(r \cdot s_P)$
5. Sys: $s := tokenSec[P]$
   if $resp_r = \mathcal{H}(r \cdot s)$ **then** $\mathcal{A}_P$ deemed authenticated

Figure 5.7: Authentication Token Using Challenge-Response

with modest computational capabilities and a keypad can play the role of B in this challenge-response protocol.

Figure 5.7 depicts such a scheme. $\mathcal{A}_P$ stores two values in some tamper-proof manner: a unique secret s_P and the hash $hpin_P$ of a PIN. Authenticating computer Sys also stores a copy of $\mathcal{A}_P$'s unique secret s_P (in $tokenSec[P]$). An unpredictable challenge r issued by Sys and secret s_P are used by $\mathcal{A}_P$ as inputs to a cryptographic hash function $\mathcal{H}(\cdot)$; the output forms the kernel of $\mathcal{A}_P$'s response. Thus, attackers, lacking knowledge of s_P, are unable to compute such responses and impersonate $\mathcal{A}_P$. And an attacker cannot steal and use $\mathcal{A}_P$ to be authenticated, because a PIN is requested and checked against $hpin_P$ before $\mathcal{A}_P$ responds to a challenge from Sys.

A variant of this approach is sometimes used by web sites where passwords are being employed to authenticate users. Users forget passwords, so some means is usually provided for a user to get another password for the web site. One common scheme works as follows.

- At enrollment, the user provides an email address which becomes part of that user's identity. The presumption is that (i) this user must successfully be authenticated on some computer in order to read email sent to that address, and (ii) that is the only way to read such email.

- The web site provides a form whereby a user can request a new password for authentication under a given identity. Submitting that form causes a password to be generated for that identity and sends email containing that password to the address stored with the identity.

In terms of the challenge-response authentication protocol sketched above, the web site takes the role of Sys, and the computer on which the user reads email takes the role of $\mathcal{A}_P$. The web site generates the response (namely, the new password) for the user rather than requiring that the user compute the response from a challenge, so there is no need for $\mathcal{A}_P$ to know a secret. And the means

by which the user authenticates to the email reader replaces the use of a PIN for authenticating the user to $\mathcal{A}_P$.

One-Time Passwords from Synchronized Clocks

Explicit challenges are not the only way $\mathcal{A}_P$ might demonstrate to *Sys* knowledge of secrets s_P and PIN_P. This subsection explores another that has been implemented in authentication tokens: transforming the current time instead of a challenge.

The basic idea is simple. To generate a one-time password, $\mathcal{A}_P$ reads a local clock and transforms the value it obtains in a way that demonstrates knowledge of s_P and PIN_P. *Sys* validates that one-time password by checking it against a value it computes by reading its local clock and estimating what time $\mathcal{A}_P$ could have read. Clock drift and message delivery delay assumptions are what enable *Sys* to estimate the clock value read by $\mathcal{A}_P$.

Figure 5.8 sketches a protocol based on these ideas. We denote the *clock* at a device D as a non-decreasing, positive function $\mathcal{C}_D(\cdot)$ of the real time (unknowable by mortals or machines) to whatever value the clock reads at that real time. Execution time of program statements is assumed to be negligible (but message delivery delay is not), and $\hat{t}$ denotes the current real time at each protocol step. So $\mathcal{C}_P(\hat{t})$ in step 3 returns the current local clock time at $\mathcal{A}_P$ when *resp* is constructed, and $\mathcal{C}_{Sys}(\hat{t})$ in step 4 returns the local clock time at *Sys* when it receives the message sent by $\mathcal{A}_P$ in step 3. The condition in the **if** statement of step 4 checks to see whether there is some candidate[22] clock time V that (i) could have been used by $\mathcal{A}_P$ in step 3 to construct a *resp* that demonstrates knowledge of s_P and PIN_P, and (ii) is larger than any clock time $\mathcal{A}_P$ already used in a valid one-time password. Together, these two conditions foil an attacker who intercepts and replays earlier valid responses from $\mathcal{A}_P$.

The amount of work required in step 4 for checking whether a response purportedly from $\mathcal{A}_P$ is valid depends on the cardinality of $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$. This, in turn, depends on two things: clock granularity, since that defines the universe of values $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ can contain, and imprecision in the estimate *Sys* has for the local clock at $\mathcal{A}_P$. Both are somewhat under our control. Clock granularity can be made coarser—for example, counting seconds instead of microseconds—although a coarser clock is not without drawbacks because, due to the $V > Last_P$ condition in step 4, only a single one-time password from $\mathcal{A}_P$ is deemed valid per tick of $\mathcal{C}_P$. And precision in the estimate that *Sys* has for $\mathcal{C}_P$ depends on the frequency of communication between $\mathcal{A}_P$ and *Sys*.

***Clock Estimation Details.** Clocks run at different rates, and therefore they can drift apart. We assume the drift rate is bounded[23] by some constant

[22] The calculation of set $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ of possible clock times used by $\mathcal{A}_P$ to construct a message received by *Sys* at local clock time $\mathcal{C}_{Sys}(\hat{t}))$ is detailed below.

[23] For clocks found in digital electronics, this value will be on the order of 10^{-6} seconds per second.

1. $\mathcal{A}_P \longrightarrow P$: "Enter PIN on keypad"
2. $P \longrightarrow \mathcal{A}_P$: PIN_P
3. $\mathcal{A}_P \longrightarrow Sys$: $\langle P, resp \rangle$ where $resp = \mathcal{H}(\mathcal{C}_P(\hat{t}) \cdot s_P \cdot PIN_P)$
4. Sys: $s := tokenSec[P]$
   $pin := userPIN[P]$
   if exists V where $V \in Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ such that
   $resp = \mathcal{H}(V \cdot s \cdot pin) \quad \wedge \quad V > Last_P$
   then
   $Last_P := V$;
   $\mathcal{A}_P$ deemed authenticated

Figure 5.8: Sketch of Time-Based Authentication Token Protocol

ρ. Thus, if for devices D and E and some real time t, we have that $\mathcal{C}_D(t) = \mathcal{C}_E(t)$ holds, then at a later real time t' we will have

$$|\mathcal{C}_D(t') - \mathcal{C}_E(t')| \leq 2\rho(t' - t). \tag{5.3}$$

The worst divergence occurs when one clock is running ρ seconds per second fast and the other ρ slow, causing them to drift apart 2ρ seconds per second.

We use intervals and some elementary interval arithmetic in the calculations that follow. For integers A and B, where $A \leq B$ holds, the notation $\Psi = [A, B]$ defines an interval Ψ comprising values v from some universe $\mathcal{U}$ of interest and satisfying $A \leq v \leq B$. When useful, we regard intervals as defining subsets of universe $\mathcal{U}$. For example, if $v \in \mathcal{U}$ holds then $v \in [A, B]$ holds if and only if $A \leq v \leq B$ holds. Given $\Psi = [A, B]$ and $\Phi = [A', B']$, define $\Psi + \Phi = [A + A', B + B']$ and $\Psi - \Phi = [A - B', B - A']$. A scalar C can be regarded as an interval $[C, C]$, and we write expressions that involve both scalars and intervals with that interpretation in mind. Thus, $C + [A, B] = [C + A, C + B]$ holds.

To construct $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$, we need a way for an authenticating computer Sys to use its local clock $\mathcal{C}_{Sys}$ and estimate local clock $\mathcal{C}_P$ at authentication token $\mathcal{A}_P$. To start, suppose Sys is given *base correction* interval Ξ_P satisfying

$$\mathcal{C}_P(t_0) \in (\mathcal{C}_{Sys}(t_0) + \Xi_P) \tag{5.4}$$

for some real time t_0. Thus, Ξ_P is a correction to clock time $\mathcal{C}_{Sys}(t_0)$ for obtaining a set of clock times that includes the clock time $\mathcal{C}_P(t_0)$ that $\mathcal{A}_P$ reads at real time t_0.

Ξ_P does not account for clock drift, so for real times t where $t_0 \leq t$ holds, Ξ_P is no longer an appropriate correction. We can account for clock drift by widening Ξ_P according to (5.3):

$$\Xi_P \quad + \quad [-2\rho(t - t_0), 2\rho(t - t_0)] \tag{5.5}$$

But without knowing real times t_0 and t, *Sys* cannot compute (5.5). *Sys* does know $\mathcal{C}_{Sys}(t_0)$ and $\mathcal{C}_{Sys}(t)$, so it can calculate $\mathcal{C}_{Sys}(t) - \mathcal{C}_{Sys}(t_0)$ as an approximation for $t - t_0$ and obtain *drift correction* interval $\Delta_P(\mathcal{C}_{Sys}(t))$ given by the following.

$$\Delta_P(\mathcal{C}_{Sys}(t)) \;=\; \Xi_P + [-2\rho(\mathcal{C}_{Sys}(t) - \mathcal{C}_{Sys}(t_0)),\ 2\rho(\mathcal{C}_{Sys}(t) - \mathcal{C}_{Sys}(t_0))\,] \quad (5.6)$$

The error in $\Delta_P(\mathcal{C}_{Sys}(t))$ compared to the interval defined by (5.5) is $\pm\rho(t-t_0)$, because we must account for $\mathcal{C}_{Sys}(\cdot)$ advancing as much as ρ faster or ρ slower than real time.

Knowledge of message delivery delays is also needed for *Sys* to calculate the set of local time values that $\mathcal{A}_P$ could have read from $\mathcal{C}_P$. These delays will typically vary, depending on the network and its load. We assume the maximum delay is known, and therefore the delay incurred by messages can be represented by an interval $\Lambda = [0, L]$ where L is the maximum clock time that can elapse on any clock while a message is in transit. Thus, a message m_T constructed and sent by $\mathcal{A}_P$ at local clock time T according to $\mathcal{C}_P$ will be received by *Sys* when $\mathcal{C}_P$ has some value in the interval $T + \Lambda$.

Suppose m_T is the message received by *Sys* in step 4 at real time $\hat{t}$. Message m_T, by the definition of Λ, would have been sent when $\mathcal{C}_P$ had some value in $\mathcal{C}_P(\hat{t}) - \Lambda$. *Sys* can estimate $\mathcal{C}_P(\hat{t})$ because, for $t_0 \le t$,

$$\mathcal{C}_P(t) \in (\mathcal{C}_{Sys}(t) + \Delta_P(\mathcal{C}_{Sys}(t)))$$

holds due to (5.6). So *Sys* can calculate set $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ of values $\mathcal{A}_P$ could have read when constructing message m_T:

$$Psbl(P, \mathcal{C}_{Sys}(\hat{t})) : \quad \mathcal{C}_{Sys}(\hat{t}) + \Delta_P(\mathcal{C}_{Sys}(\hat{t})) - \Lambda \quad (5.7)$$

The cardinality of $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ depends on the widths of intervals Λ and $\Delta_P(\mathcal{C}_{Sys}(\hat{t}))$. Unlike Λ, which has fixed width, $\Delta_P(\mathcal{C}_{Sys}(\hat{t}))$ grows ever wider due to (5.6) as time increases; left unchecked, this growth will eventually dominate. And this growth is problematic because the amount of work required by the **if** in step 4 of Figure 5.8 is proportional to the cardinality of $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$.

The ever increasing width of $\Delta_P(\mathcal{C}_{Sys}(\hat{t}))$ accounts for possible divergence of $\mathcal{C}_{Sys}$ and $\mathcal{C}_P$ due to drift. However, message m_T received by *Sys* at clock time $\mathcal{C}_{Sys}(\hat{t})$ and containing a valid one-time password can enable *Sys* to refine its estimate of $\mathcal{C}_P$ and reduce the width of $\Delta_P(\mathcal{C}_{Sys}(\hat{t}))$ accordingly. In particular, we above argued that $\mathcal{C}_P(\hat{t})$ equals some value in the interval $T + \Lambda$ at real time $\hat{t}$ that m_T is received by *Sys*, so we can define a *revised* base correction interval

$$\Xi_P = \mathcal{C}_{Sys}(\hat{t}) - (T + \Lambda) \quad (5.8)$$

which captures the difference $\mathcal{C}_{Sys}(\hat{t}) - T$ between the clocks at *Sys* and $\mathcal{A}_P$ at the time m_T is sent[24] in addition to uncertainty Λ in message delivery delay. The width of Ξ_P is now the width of Λ. Moreover, Ξ_P defined by (5.8) satisfies

$$\mathcal{C}_P(\hat{t}) \in (\mathcal{C}_{Sys}(\hat{t}) + \Xi_P),$$

[24]This assumes an inconsequential relative clock drift over the period m_T is in transit.

which is (5.4) with t_0 replaced by the time $\hat{t}$ that m_T is received by *Sys*. This means that by using revised base correction interval Ξ_P in (5.6), we can replace t_0 there by $\hat{t}$ and reduce the width of interval $\Delta_P(\mathcal{C}_{Sys}(\hat{t}))$ to Λ, thereby eliminating accumulated unnecessary compensation for clock drift.

We have thus established that the calculation of $Psbl(P, \mathcal{C}_{Sys}(\hat{t}))$ in the protocol of Figure 5.8 should be done according to (5.7), where Δ_P is defined by (5.6) in terms of a revised base correction interval Ξ_P, which is calculated according to (i) the clock time used by $\mathcal{A}_P$ in computing the last valid password *Sys* received from $\mathcal{A}_P$, (ii) the local clock time at *Sys* when that message was received, and (iii) the variance in delivery delay. This calculation can be performed by employing an interval-valued program variable Ξ_P and recording in another new variable $Last^P_{Sys}$ the local time at *Sys* used in calculating Ξ_P. We augment Figure 5.8 by inserting assignment statements

$$\begin{aligned} &Last^P_{Sys} := \mathcal{C}_{Sys}(\hat{t}); \\ &\Xi_P := Last^P_{Sys} - (Last_P + \Lambda) \end{aligned}$$

just after the assignment to $Last_P$ in step 4, and use the following definition of $\Delta_P(\mathcal{C}_{Sys}(t))$:

$$\Delta_P(\mathcal{C}_{Sys}(t)) \;=\; \Xi_P + [-2\rho(\mathcal{C}_{Sys}(t) - Last^P_{Sys}),\ 2\rho(\mathcal{C}_{Sys}(t) - Last^P_{Sys})\,]$$

New program variable Ξ_P must also be initialized. This can be done by bringing $\mathcal{A}_P$ and *Sys* in close proximity. $\mathcal{C}_P$ can now be adjusted so that it reads approximately the same value as $\mathcal{C}_{Sys}$; Ξ_P is then set equal to $[0, \epsilon]$, where ϵ is an upper bound on the error in this clock synchronization procedure.

RSA SecurID. SecurID, developed by RSA Security, is an authentication token that implements one-time password generation using a protocol similar to that just outlined. It enjoys wide-spread acceptance in the market as a way for people at remote computers to authenticate themselves to some central computing facility.

The SecurID token itself is sized to fit on a key ring. It has an LCD display but no keyboard. And it is built to be tamper-resistant, so an attacker who steals a SecurID token is unlikely to be able to physically open it and learn the secret it stores. Inside each SecurID token is a clock, circuits for computing a proprietary cryptographic hash function, and storage for a unique 128 bit secret installed at the factory. The electronics are battery powered, with the expectation that the entire token will be replaced every 3 years.

The most significant difference between the protocol given in Figure 5.8 and what SecurID implements comes from the absence of a keypad on the SecurID token. The keypad on the user's computer is used instead, and the computation of $\mathcal{H}(\mathcal{C}_P(\hat{t}) \cdot s_P \cdot PIN_P)$ in step 3 of Figure 5.8 is split between the SecurID token and software installed on the user's (remote) computer. Specifically, a SecurID token $\mathcal{A}_P$ constantly displays $\mathcal{H}_1(\mathcal{C}_P(\hat{t}), s_P)$ on its LCD, where $\mathcal{H}_1(\cdot)$ is a proprietary cryptographic hash function. A user P, wishing to be authenticated

1. $\mathcal{A}_P \longrightarrow Sys$: P requests authentication
2. $Sys \longrightarrow \mathcal{A}_P$: $iSysP$ where $iSysP = i^P_{Sys}$
3. $\mathcal{A}_P$: $i_P := \max(i_P, iSysP) + 1$
   if $i_P > N$ **then halt**(Password list exhausted!)
   else $\mathcal{A}_P \longrightarrow Sys$: $\langle P, iP, resp \rangle$
   where $iP = i_P, resp = M_P[i_P]$
4. Sys: **if** $i^P_{Sys} < iP \wedge M^P_{Sys}[iP] = resp$ **then**
   $i^P_{Sys} := iP$
   $\mathcal{A}_P$ deemed authenticated

Figure 5.9: One-time Password List Authentication

at his computer, starts running an application on that machine; it prompts the user for a PIN PIN_P and for the value being displayed on $\mathcal{A}_P$'s LCD. The application combines these using a second hash function $\mathcal{H}_2$ and obtains the kernel of its response $resp = \mathcal{H}_2(PIN_P, \mathcal{H}_1(\mathcal{C}_P(\hat{t}), s_P))$, which is then sent to the central computing facility. The first conjunct of Step 4 in Figure 5.8 must be revised to accomodate this slightly different construction of *resp*.

One-Time Passwords from Hash Chains

In the schemes discussed so far, the sequence of one-time passwords is not pre-computed by an authentication token; rather, it depends on a sequence unpredictable challenges (Figure 5.7) or depends on the times that the one-time passwords are generated (Figure 5.8). In this section, we explore a scheme where the sequence of one-time passwords can be pre-computed. The scheme admits manual and electronic implementations, ranging from lists recorded on paper to cryptographic calculators to full-blown processors having long-term storage.

To start, we suppose $\mathcal{A}_P$ stores an array $M_P[1..N]$ containing N one-time passwords m_1, m_2, ..., m_N, and a counter i_P (initially 0). Authenticating computer *Sys* stores its own copy M^P_{Sys} of M_P, and a counter i^P_{Sys} (initially 0). These variables will satisfy the following:

$$\text{For } 1 \leq i \leq N : M_P[i] = M^P_{Sys}[i] = m_i. \tag{5.9}$$

$$\text{For } i_P < i \leq N : m_i \text{ has not been sent by } \mathcal{A}_P \text{ to } Sys. \tag{5.10}$$

$$\text{For } i^P_{Sys} < i \leq N : m_i \text{ has not been received by } Sys \text{ from } \mathcal{A}_P. \tag{5.11}$$

If $\max(i_P, i^P_{Sys}) < i$ holds and the list of one-time passwords is initially secret from attackers then $M_P[i]$ can serve as a one-time password without fear that it has been seen by a wiretapper. The protocol is given in Figure 5.9.

Condition $i^P_{Sys} < iP$ in step 4 prevents replay attacks from succeeding. Moreover, by requiring that $i^P_{Sys} < iP$ hold rather than stronger condition

$i^P_{Sys} + 1 = iP$, the protocol is able to tolerate lost messages between $\mathcal{A}_P$ and *Sys*. Retransmission is not a suitable remedy for lost messages sent by $\mathcal{A}_P$ in step 3, because *Sys* has no way to distinguish duplicate messages sent by $\mathcal{A}_P$ from replay attacks.

Notice that $\mathcal{A}_P$ can be implemented with an ordinary page of paper containing numbered lines, with $M_P[i]$ appearing on the line numbered i. Whenever a password is sent to *Sys*, the corresponding line is crossed out, effectively storing the current value of i_P on the page.

The assumption that password list M_P is finite has a price. Any finite-length list of one-time passwords will be exhausted sooner or later. Some protocol is then needed to refresh the one-time password list by storing new values in M_P and M^P_{Sys}. The obvious protocol requires authenticated and confidential communication between $\mathcal{A}_P$ and *Sys*. Such communication is easily supported if $\mathcal{A}_P$ and *Sys* are in close physical proximity. A person P who is in the office even for brief periods could, just before leaving, print a new page of passwords or could refresh $\mathcal{A}_P$ by connecting it to the corporate intranet. The person who is away for long periods might not have such opportunities often enough, though.

Lamport's Hash Chain Scheme. Instead of precomputing and storing the sequence of one-time passwords, each element can be calculated on demand. Let $\mathcal{H}(\cdot)$ be a cryptographic hash function. The sequence obtained by defining $m_{i+1} = \mathcal{H}(m_i)$ does not work because it allows attackers to compute unused one-time passwords from used ones. But that vulnerability does not exist if we build the chain in the other direction, defining $m_i = \mathcal{H}(m_{i+1})$. So we choose a random (secret) value s_P, set $m_N = \mathcal{H}(s_P)$, and set $m_i = \mathcal{H}(m_{i+1})$ for $1 \leq i < N$. The resulting one-time password sequence is[25]

$$\begin{array}{cccccc} \mathcal{H}^N(s_P), & \mathcal{H}^{N-1}(s_P), & \ldots & \mathcal{H}^2(s_P), & \mathcal{H}^1(s_P) \\ m_1, & m_2, & \ldots & m_{N-1}, & m_N \end{array} \tag{5.12}$$

which we can succinctly characterize by:

$$\text{For } 1 \leq i \leq N : m_i = \mathcal{H}^{N-i+1}(s_P) \tag{5.13}$$

Moreover, from collision resistence of cryptographic hash functions, we have for $\mathcal{H}(\cdot)$ and all i and j satisfying $i \leq j$:

$$\mathcal{H}^{j-i}(x) = m_i \;\; \Rightarrow \;\; x = m_j \tag{5.14}$$

Sys decides if a one-time password *resp* received from $\mathcal{A}_P$ is valid by determining whether

$$(\exists\, i > i^P_{Sys} :\; m_i = resp) \tag{5.15}$$

[25] Superscripts on function names denote nested function application: $F^{i+1}(x) = F(F^i(x))$ and $F^0(x) = x$.

1. $\mathcal{A}_P \longrightarrow Sys$: P requests authentication
2. $Sys \longrightarrow \mathcal{A}_P$: $iSysP$ where $iSysP = i^P_{Sys}$
3. $\mathcal{A}_P$: $i_P := \max(i_P, iSysP) + 1$
   if $i_P > N$ **then halt**(Password list exhausted!)
   else $\mathcal{A}_P \longrightarrow Sys$: $\langle P, iP, resp \rangle$
   where $iP = i_P, resp = \mathcal{H}^{N-i_P+1}(s_P)$
4. Sys: **if** $i^P_{Sys} < iP \wedge \mathcal{H}^{i_P - i^P_{Sys}}(resp) = last_P$ **then**
   $i^P_{Sys} := iP$
   $last_P := resp$
   $\mathcal{A}_P$ deemed authenticated

Figure 5.10: Hash Chain Authentication Protocol

holds, since (5.11) implies that m_i is fresh, hence valid, if $i > i^P_{Sys}$ holds. And Sys can determine that (5.15) holds by storing $last_P$ satisfying[26] $last_P = m_{i^P_{Sys}}$ and checking whether

$$iP > i^P_{Sys} \ \wedge \ \mathcal{H}^{iP - i^P_{Sys}}(resp) = last_P \tag{5.16}$$

holds, based on the following reasoning:

$$\begin{aligned}
& iP > i^P_{Sys} \ \wedge \ \mathcal{H}^{iP - i^P_{Sys}}(resp) = last_P \\
= \ & iP > i^P_{Sys} \ \wedge \ \mathcal{H}^{iP - i^P_{Sys}}(resp) = m_{i^P_{Sys}} \\
\Rightarrow \ & iP > i^P_{Sys} \ \wedge \ m_{iP} = resp \\
\Rightarrow \ & (\exists\, i > i^P_{Sys} : \ m_i = resp)
\end{aligned}$$

Notice that with this freshness test, there is no need for Sys to store s_P or anything from which s_P can be derived, nor is there any need for Sys to store the sequence of one-time passwords.

After replacing references to M_P and M^P_{Sys} in Figure 5.9 with corresponding expressions in terms of $\mathcal{H}(\cdot)$, we obtain the protocol of Figure 5.10. In step 3, $resp$ could be obtained by calculating $\mathcal{H}^{N-i_P+1}(s_P)$ but the alternative (discussed above) of $\mathcal{A}_P$ using the correct value from a stored list $\mathcal{H}^N(s_P)$, $\mathcal{H}^{N-1}(s_P)$, ..., $\mathcal{H}^1(s_P)$ would also work. Calculation of $\mathcal{H}^{N-i_P+1}(s_P)$ is straightforward if $\mathcal{A}_P$ is a digital device; simply copying the the appropriate element would be the more sensible course when $\mathcal{A}_P$ is implemented by a piece of paper.

Use of a hash chain for generating a list of one-time passwords does introduce a vulnerability, however. Since $\mathcal{H}(\cdot)$ is public, an attacker who learns a valid but not yet used one-time password (say) m_F is able to compute all earlier

[26] Define $m_0 = \mathcal{H}^{N+1}(s_P)$ so that $last_P$ can be initialized (since initially $i^P_{Sys} = 0$).

passwords m_{F-i} for $1 \leq i < F$ by using:

$$\text{For } 1 \leq j < i \leq N : m_{i-j} = \mathcal{H}^j(m_i)$$

And among those, the m_k satisfying $i^P_{Sys} < k \leq F$ are valid one-time passwords because they have not yet been accepted by *Sys*. So the m_k could be used by an attacker to impersonate $\mathcal{A}_P$.

Moreover, there are circumstances that could allow an attacker to learn m_F. If $\mathcal{A}_P$ is not authenticating the source of the challenge (of step 2) that engenders the response it sends in step 3 of Figure 5.10, then an attacker could impersonate *Sys* and send $F-1$ to $\mathcal{A}_P$ as challenge *iSysP*. $\mathcal{A}_P$ would then respond to the attacker with $\langle P, F, m_F \rangle$. There are two defenses against this so-called "large challenge attack". One is for $\mathcal{A}_P$ to authenticate the source of a challenge before responding to it. The second, which does not require authenticating the source of a challenge, is for $\mathcal{A}_P$ to retain the last challenge it receives and to ignore any subsequent challenge that is much larger, hence potentially constitutes a "large challenge attack".[27]

S/KEY Implementation. The protocol of Figure 5.10 is the basis for the S/KEY one-time password authentication scheme, which allows remote users to be authenticated by a central server. S/KEY one-time passwords are 64 bits long. They are generated using a hash chain built with MD4, where each 128 bit MD4 output (i.e., 16 bytes) is transformed into the 64 bits desired password by XORing byte 1 with byte 2, byte 3 with byte 4, ..., and byte 7 with byte 8.

The secret s_P being repeatedly hashed in S/KEY is the concatenation of a secret passphrase user P chooses and a *per-user seed* the central server includes with each challenge. Thus instead of storing s_P, authetication token $\mathcal{A}_P$ prompts the user for the passphrase and then uses the per-user seed to calculate s_P. Per-user seeds are different for different users and different servers. So three things, none of which needs to be kept secret, are being stored for each user P by a server: the per-user seed, $last_P$, and i^P_{Sys}. Per-user seed values that differ on the different servers enable a user to employ the same passphrase for authenticating to different systems, since different hash chains will now be generated for each different system. With the per-user seed a part of s_P, we also get a simple way for switching to a new sequence of one-time passwords once the old sequence has been exhausted—the server increments the per-user seed and both i_P and i^P_{Sys} are reset to 1.

Sympathetic to the problems of transcribing 64 bit numbers by hand, S/KEY's designers provided a dictionary of 2048 short English words and a fixed correspondence between dictionary entries and bit strings. Each 64 bit S/KEY one-time password thus corresponds to a sequence of these dictionary words. Users who manually perform the authentication token's protocol for S/KEY (perhaps

[27] Significantly larger challenges are not necessarily symptoms of an attack. They also arise if the communication link between *Sys* and $\mathcal{A}_P$ is unreliable. Even in that case, though, system administrators ought to be alerted since there is a problem.

with the help of a page that lists the hash chain) interact with S/KEY using one-time passwords represented in this easier-to-manipulate encoded form; software implementations use raw 64 bit values.

5.3 Something You Are

Humans virtually never authenticate each other based on what we know or have. Instead, we use *biometrics*—physical or behavioral traits—exhibited by the person being authenticated. We recognize acquaintances by their faces or voices, and police use fingerprints or DNA to place suspects at a crime scene. Something you are is the essence of this approach to authentication.

Biometric authentication is seen as attractive for a variety of reasons. There is nothing to forget, nothing to lose, nothing to protect against theft, and nothing that can be shared. Moreover, certain biometrics (e.g., face recognition) offer the promise of identifying individuals without their knowledge, which would be useful for screening crowds in public places.

Not all biometrics are suitable for authentication, though. For authenticating individuals in some population, a biometric must satisfy certain requirements.

> **Biometrics for Authentication.** The biometric must be distinct and invariant over time for each member of the population. It must be difficult to spoof, yet relatively easy to measure. And individuals must be willing to subject themselves to the measurement process. □

These requirements imply that biometric authentication almost certainly will need to be augmented with an alternative. Some people are afraid of sensors that require peering into an eyepiece or placing a finger into a measurement device; others are hesitant to touch any sensor that, because others touch it, might harbor bacteria. Missing or deformed body parts mean that specific biometrics might not be available for everyone, while environmental and occupational exposure can render biometrics difficult to sense reliably. And various religions prohibit their followers from having certain biometrics measured.[28]

A biometric authentication system can admit two kinds of errors. It can fail to authenticate an individual it should, or it can mistakenly authenticate an individual it shouldn't. The *false reject rate* (FRR) gives the likelihood of committing the first kind of error, and the *false accept rate* (FAR) gives the likelihood of committing the second kind.[29] False rejects typically arise from errors

[28] For example, old order Amish and Mennonites hold that the second commandment ("You shall not make for yourself a graven image, or any likeness of anything that is in heaven above, or that is in the earth beneath, or that is in the water under the earth..." Exodus 20:3-6) prohibits making images of their followers. Muslims too are enjoined from making images of animate beings. And certain Orthodox Jews interpret the Shulchan Aruch (chapter 11) as prohibiting the creation of a person's image except for purposes of study, though the more pervasive view in Judaism is that only images that could be the object of worship (i.e., idols) are prohibited.

[29] False rejects are sometimes called *type I errors* and false accepts are called *type II errors*,

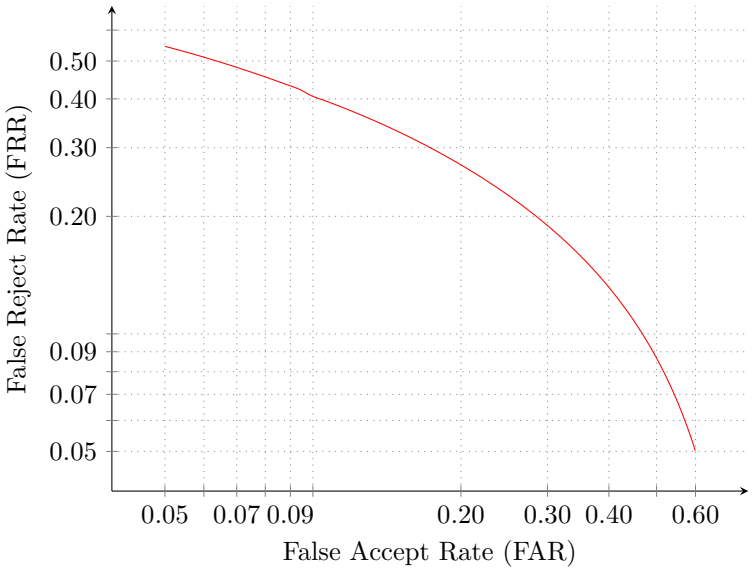

Figure 5.11: Example DET Curve

in sensing a biometric. Such errors are inevitable when measuring an analog physical property in an uncontrolled and dynamic environment, especially since physiological characteristics that vary over time (from stress, illness, or other natural phenomena) might affect what is being measured.

False rejects can be reduced by allowing inexact matches between what is sensed when authentication is attempted and what was sensed at enrollment. But false accepts will increase as the tolerance for these inexact matches becomes more lax. Thus, decreases in FRR are typically accompanied by increases in FAR. The trade-off between FRR and FAR is summarized as a *detection error trade-off* (DET) curve, where the x axis is the FAR and the y axis is the corresponding FRR. DET curves are typically non-linear, as illustrated in Figure 5.11.

The designer of a biometric authentication system must select some operating point on the DET curve. One strategy is to choose a point that is close to the origin, arguing that this best approximates a system where both FRR and FAR equal 0. However, this strategy presumes that the penalty for false rejects is the same as the penalty for false accepts. It rarely is. So the more sensible course is to decide which of false rejects or false accepts is the more problematic, and then choose an acceptable rate for that kind of error (ignoring the other kind). As an example, we would want to minimize false accepts for authenticating people who access a top-secret government facility and, therefore, we would

thereby taking "the human being authenticated has the identity being claimed" as the null hypothesis.

choose a small value for FAR.[30] But minimizing false rejects is probably the better course for admittance to a members-only health club, because turning away a member causes embarrassment, hence is bad for business, even though implementing a low FRR occasionally allows non-members entry.

5.3.1 Implementing Biometric Authentication

A typical biometric authentication system involves

- a front-end subsystem that senses a biometric and converts it to some digital form, and
- a back-end system that processes this digital data and compares it against a *template* stored for the individual being authenticated.

Templates are designed to facilitate inexact matching, and therefore they typiclly record only an approximation of a subject's biometric. For example, rather than storing the image of a fingerprint, a template might contain only certain essential characteristics, such as the (relative) locations of the start, end, or crossings of a few prominent ridges in that fingerprint. The front-end system would sense the surface contours of a finger; the back-end system would use signal processing algorithms to identify ridges from the surface contours and determine whether these features are consistent with a given template.

Users must enroll before being eligible for authentication. At enrollment, the system administrator gives a unique identifier to the user; the user then enters that unique identifier and allows the front-end subsystem's sensors to sample the biometric. A template for that user's identifier is synthesized from the samples. Thereafter, to be authenticated, a user enters an identifier and the front-end subsystem's sensors samples the user's biometric. That biometric data is then checked for a sufficiently good match with the template for the identifier just entered.[31]

A front-end subsystem must defend against spoofing, where one person provides bogus data to the biometric sensor in an attempt to be authenticated as another. These attacks often involve physical artifacts that replicate features of the person being impersonated.[32] The defense against such attacks is to ascertain that part of a live human is measured by the front-end system. To test for liveness, the front-end system might check for involuntary signals generated by a living body (e.g., pulse, body temperature, brain waves) or measure a response to a stimulus (such as pupil dilation when light intensity is increased). But a

[30] The system might accommodate the corresponding large number of false rejects by having some additional means of authentication. Human guards at the gate could implement that.

[31] For identification based on a biometric, no identifier is provided, and the back-end subsystem searches all of the templates for (inexact) matches.

[32] In the case of fingerprints, for example, at attacker might attempt to masquerade as a person P by (i) inserting a facsimile of P's finger into the fingerprint reader, (ii) wearing thin rubber gloves (or applying rubber pads) molded to have P's fingerprints, or even (iii) cutting-off P's finger or hand and using that. Moreover, attacks (i) and (ii) are not only theoretical possibilities but have been successfully demonstrated.

simple and effective defense against most kinds of spoofing is simply to have a human guard supervise data capture by the biometric sensor.

A biometric authentication system, like any other authentication system having a component that matches a response with some stored data, must also defend against attacks that attempt to replace one or both inputs to the matching algorithm. The channel that links the front-end system with the back-end system must be secure, because an attacker who can alter data enroute can impersonate an individual by putting that person's biometric sensor values into the message the back-end system receives. And the integrity of stored templates must be protected, to prevent replacing a *bona fide* user's biometric characteristics with those of an attacker.

5.3.2 A Catalog of Biometrics

We catalog below various biometrics in use today to authenticate humans. The biometrics differ not only in how well they discriminate between subjects but also in other ways that, depending on the application, could be important: cost, how much cooperation by a subject is required for sensing the biometric, and whether subjects are even aware the biometric is being measured.

Fingerprints. These are characterized by what are called *minutiae*—features of the raised ridges that appear in the skin on human fingertips. Beginnings, endings, and bifurcations of ridges are what is usually matched, where a match involves checking the parameters that give the feature's relative location in the plane and its angle of orientation to the ridge.

The existence today of reliable, low-cost fingerprint readers means that biometric fingerprint sensors are increasingly found on systems, such as laptops and point of sale terminals, where cost matters and an alternative to passwords is sought. Fingerprint sensors on laptops are typically not supervised and do not implement liveness tests, so these are vulnerable to spoofing. Customer use of point of sale terminals is likely supervised by a sales clerk but a clerk's use might not be, so spoofing by clerks could be a concern. Over 2% of human subjects do not have fingerprints that can be reliably measured.

Faces. Absolute dimensions as well as the proportions of a face and its features are the match criteria here.[33] Video cameras enable measurement to be done at a distance and without the cooperation or knowledge of subjects. Some systems work by finding specific facial features (often by first locating the eyes) and determining matches by comparing the relative location, size, and general shape of each feature (e.g., nose, mouth, and jaw edges) to what is recorded in a template. Other systems use neural networks or other statistical learning techniques that have been trained to match faces. A third family of systems is

[33] Facial tomography, which images body heat distribution over the subject's face, has also been proposed. Such images, however, are not invariant, being affected by alcohol, stress, and the subject's overall medical condition.

based on decomposing a face into a weighted sum of elements from a relatively small set of archetypal face images called *eigenfaces*; a template stores only the weights, and a face is deemed to match the template if its decomposition involves similar weights for each component eigenface.

Commercial face recognition systems that use frontal images to authenticate cooperating subjects work reasonably well. But face recognition for surveillance—identification without a subject's cooperation—do not yet work well because field of view, background, lighting, and the subject's pose make it difficult to locate and separate a face from the rest of a scene. Facial expression, aging, and disguises (including facial hair, glasses, or cosmetics) create additional challenges for establishing a match.

Eyes. The iris and the retina in human eyes have each been used for biometric authentication.

Iris recognition is based on the pattern of pigments in the ring of colored tissue that surrounds the pupil. This pattern is believed to be unique for each eye and person; it stabilizes once a person has reached adolescence. Sensors can image an iris from as far as 2 feet away without subject-positioning devices (such as an eye piece or chin rest), so few are uncomfortable with the measurement process. Either hippus[34] or the pupil's reaction to changes in illumination can serve as a liveness test. Iris recognition systems have been used in airports—domestically and abroad—to streamline flight boarding by frequent flyers and for immigration.

The retina is the back surface of the eyeball. By projecting a low-intensity beam of light, the unique pattern of veins beneath the retinal surface can be recorded and used for authentication. This process requires the subject to focus on a point while looking through an eyepiece. Although some find that measurement process off-putting, FAR and FRR are considerably better than what iris recognition achieves (and the performance of iris recognition dominates all other biometrics in our catalog). Inconvenience limits the deployment of retina scanning to high-security settings, where compromise in favor of greater security is accepted because it is part of the culture.

Hands. A sensor here images a silhouette of the hand; measurements of the palm as well as length, width, and thickness of the four fingers are then used as a basis for matching. This biometric is largely unaffected by scars, ridges and tattoos on the hand, but large rings, bandages, and gloves can lead to errors.

Even though the approach is not very discriminating (with scant data published about the FAR and FRR for deployed systems), hand-recognition systems today enjoy success in at least one niche market: Disney World in Orlando, Florida. Here, hand recognition is employed for authenticating users of season passes. This prevents tour operators or local residents from purchasing a season pass and loaning it to a succession of different visitors who would otherwise have bought more-expensive short-term passes. Enrollment is performed

[34] Hippus is the normal .5 Hz oscillation of the pupil in a human eye.

the first time the season pass is used; subsequent uses of the pass require the subject's hand to match the template created at enrollment.[35]

Impracticable Prospects

Absent from the above catalog are those biometrics believed unlikely to be deployable in the foreseeable future. In some cases, suitable sensor and matching technology does not yet exist; in others, the requirements of distinctness or invariance are not likely to be met. These impracticable biometrics include:

- Handwritten signatures, whether measured by final appearance or by capturing the series of accelerations (or forces) that produced the signature. Variation in the signatures produced by an individual is too high.
- Voice verification, where tone, pitch, and resonance are measured for a human's vocalization of some system-provided challenge phrase (as compared to always uttering the same phrase, which would be vulnerable to spoofing using "playback" attacks). Sufficient discrimination cannot be achieved because matching must account for variations due to the speaker's health (colds and sore throats) and the environment (including the exact microphone, ambient noise, and room acoustics).
- Body odor, because although dogs successfully use this to identify people (or drugs and other contraband), good sensors do not yet exist to convert odors into electronic signals—especially taking into account the ways human odor is affected by diet, emotional state, and is masked by deodorants.
- Brain wave measurements of how a subject reacts to seeing familiar images as compared with an unfamiliar images, thereby implementing a form of knowledge-based authentication. The problem here is that an attacker can spoof by becoming familiar with images that should be known to the subject being impersonated.

5.4 Privacy Pitfalls of Authentication

Authentication, when undertaken injudiciously, can lead to privacy violations. First, in authenticating somebody, you learn an identity and thus, by definition, you learn an associated set of attributes. Some of these attributes might be considered personal information. So authentication can cause personal information to be revealed.

A second threat to privacy arises when authentication is used to validate who participates in some action. If such participation is deemed private (as, for

[35] Disney also sells sets of linked season passes, where a subject is deemed authenticated if his hand matches the template associated with any season pass in the set. This means season passes in a linked set are interchangeable, which enables a family to purchase a set of season passes without requiring them to keep track of who was the first to use each pass in that set.

example, certain purchases or medical procedures could be), then a side effect of authentication is to associate personal information with an identity. This problem is compounded when the same identifier is authenticated in connection with multiple actions. For example, the use of social security numbers as a quasi-universal identifier in the U.S. eroded privacy by allowing third parties to connect seemingly unrelated actions with a single individual and then to make inferences that associated additional attributes with that individual.

Requiring authentication can also have a chilling effect. The prospect of undergoing authentication inhibits people from engaging in activities they fear could be misconstrued, deemed inappropriate, or lead to retribution. So in requiring authentication, we institute a *de facto* form of authorization. Of concern (here) is not that basic freedoms might be eroded when authentication is required but that this erosion is inadvertent. The concern is that policy—not side-effects of a system's construction—should be what dictates who may engage in what activities, and authorization—not authentication—mechanisms should be what implements such policy.

Finally, it is worth noting that information not collected cannot be abused. Authentication collects information and possibly even stores it for subsequent use. Widespread deployment of authentication mechanisms thus increases the chances for privacy violations in three ways. First, personal information could be abused by the agency collecting it. Second, stored personal information could be stolen. And finally, having the personal information further increases the risk of inferences by linking shared identities or other shared attributes.

5.4.1 Guidelines for Deploying Authentication

The above privacy pitfalls can be avoided by following four simple guidelines concerning the use and implementation of authentication mechanisms:

> **Seek Consent.** Authenticate people only with their consent, and inform them if information about an identity will be saved. People using a system thus become aware that they are relinquishing some control over the confidentiality of personal information in that identity. □

> **Select Minimal Identity.** Authenticate only against identities that embody the smallest set of attributes needed for the task at hand. Personal information is thus not disclosed unnecessarily. □

> **Limit Storage.** Do not save information about authenticated identities unless there is a clear need, and delete that information once it is no longer needed. This reduces the chances that saved identity information can subsequently be re-targeted for uses not implied by the consent of the user that allowed its collection. □

> **Avoid Linking.** Eschew including the same unique attribute (e.g., identifier) in different identities. A single, shared attribute allows linking the identities that contain this attribute, and that could violate privacy by

revealing attributes comprising one identity to those who learn the other identity. □

To see these guidelines in action, consider a magnetic-card scheme for limiting off-hours building-access to members of a community. With the system implemented at Cornell, each student and staff member is issued a Cornell University id card. A unique id number, which serves as an identifier for the card owner, is encoded on the card's magnetic stripe. Magnetic-card readers situated outside building doors are connected to computers, which in turn are connected to actuators that control the door locks. Some of these computing systems log all building-access attempts. So revisiting the above guidelines, we have:

Seek Consent. Since one must take an action—place a card in a reader—to enter a building, authentication is being done with consent. However, no notice is being given about whether access attempts are being logged by the computers.

Select Minimal Identity. We might debate whether the appropriate identity is being used for the authentication. Membership in the Cornell community is the sole attribute needed to enter most university buildings, so an id card stating only that attribute should suffice there. Since the id card in use encodes a unique id number for the card holder, more information is being disclosed than necessary.

Limit Storage. Should the system log the unique id number and time of day whenever a card holder enters a building? Such information might be useful if a crime is committed in that building, because the log identifies people that law enforcement officers might want to interview for clues of unusual activity. However, this benefit must be weighed against the errosion of privacy and the potential for abuse that become possible once a building-entry log exists, since an individual's comings and goings can now be tracked.

Avoid Linking. Cornell University id cards are used in connection with a wide range of activities, such as borrowing books from the library, parking a car in the university garage, and charging food at university-run cafeterias. The unique id number on each id card thus enables linking an individual's actions, unnecessarily eroding that person's privacy.

5.4.2 Privacy Pitfalls for Specific Technologies

Authentication methods based on "something you have" or "something you are" are particularly insidious. First, many of them can proceed without a subject's knowledge or involvement; they not only violate the Seek Consent guideline but (as we show below) can be leveraged to reveal personal information. Second, some of the biometrics that uniquely characterize subjects are inextricably linked to personal information; the Select Minimal Identity guideline is then violated.

RFID Chips. Today's RFID chips, for the most part, are indiscriminate about responding to interrogation signals. This functionality is invaluable for identifying and inventorying objects in a store or warehouse; and Seek Consent is not being violated, because that guideline does not apply to inanimate objects. But Seek Consent does apply to people, so indiscriminate responses become problematic when RFID chips are used for authenticating humans. The obvious solution is to provide human subjects with some way to inhibit an RFID chip's response to an interrogation signal. For example, newer United States passports have an embedded RFID chip, and the passport cover incorporates a layer of foil which implements a Faraday cage. The passport cover must be open before the RFID chip can receive an interrogation signal; opening your passport cover signifies your consent to be authenticated.

Even when RFID chips are limited to authenticating inanimate objects, privacy problems can still arise. Consider an inanimate object O that is known (with high probability) to be carried by a specific individual S. By interrogating O, a system then (with high probability) authenticates S. But S received no indication of being authenticated and had no opportunity to give consent. The problem here is a form of linking—linking based on physical proximity rather than common attributes that could be deleted from the identity of S or O. Such *proximity linking* can be prevented if S or some other responsible party is able to deactivate any RFID chips embedded in objects S carries. For example, a store that monitors its inventory and implements item-pricing by having an RFID chip embedded in each item it sells arguably should deactivate that chip at the check-out counter when taking the customer's payment.[36]

One final kind of privacy violation becomes possible when (i) the very possession of an object O is considered personal information by S and (ii) O contains an RFID chip that indiscriminately responds to interrogation signals. For example, O might be a book about a subject considered subversive; and the embedded RFID chip response might contain the title or other particulars of the book. But even when RFID chips are embedded in mundane objects, new vulnerabilities can be created. An attacker could determine the contents of a house from the street simply by broadcasting interrogation signals and then interpreting the responses; a business could assess your taste and spending levels by interrogating the RFID chips in your clothing as you enter. All of these are instances of proximity linking and, as before, the solution is to know about and be able to deactivate RFID chips in objects. In short, a safe strategy is for loquacious RFID chips to be silenced.

[36] Deactivation does deprive the customer of any benefits embedded RFID chips can provide. For example, by embedding an RFID chip in the packaging for a perishable food, we can provide a purchaser with the history of temperatures to which that food has exposed; an RFID in the shipping container for a fragile object could log exposure to shocks, which would be useful in resolving disputes with shippers about breakage. Instructions for use or details about an item's pedigree, components, or warranty, which are useful later in an object's life, become virtually impossible to misplace if they are stored by an RFID embedded in the object itself.

Biometrics. Beyond giving a basis for recognition, a biometric necessarily conveys something about the physical attributes of the individual being recognized. Such attributes might well be considered personal; if they are, then using the biometric can violate an individual's privacy. The challenge when deciding whether or not to use some candidate biometric for authentication, then, is to ascertain (i) what attributes can be deduced from the candidate biometric and (ii) whether they will be considered personal.

Deoxyribonucleic acid (DNA) is perhaps the extreme. Your DNA not only distinguishes you, but it contains a genetic code that reveals information about medical conditions you could develop and, because it incorporates the DNA from your progenitors, reveals information about your "blood" relatives too. Most people would regard such information as highly personal. For instance, a higher susceptibility to cancer is something you wouldn't want a potential employer to know, because it portends increased absence from work and perhaps increased medical costs, which make you a less attractive candidate.

Biometrics in common use today are nowhere near as revealing as DNA, but nevertheless today's biometrics can violate privacy. Voice pitch and many body features are predicted by gender; fingerprints reveal information about occupation; and certain facial characteristics are correlated with race. You might not know with certainty whether an individual is a white caucasian male, but certain biometrics could suggest it likely—and in some settings, having that knowledge would be inappropriate.

5.4.3 Identity Fraud and Identity Theft

In *identity fraud*, a criminal obtains information about a victim and uses this information to impersonate and charge purchases to that victim's accounts; with *identity theft*, the criminal initiates new business relationships (unknown to the victim) and uses them for purchases or other actions attributed to the victim. Both forms of *identity crime* benefit from the deployment of weak methods to authenticate customers, so attackers are able to spoof the authentication mechanism and thereafter engage in transactions that will be (mis)attributed to *bona fide* customers.

The current epidemic of identity crime can be directly attributed to the widespread use of knowledge-based authentication employing queries with well-known answers. This approach to authentication minimizes any inconvenience experienced by customers, who are now not burdened with memorizing a secret or carrying an authentication token. But now anyone who learns an individual's identifiers (e.g., social security number, credit-card number, etc.) and a few other widely known attributes (e.g., home phone number or mother's maiden name) can impersonate that individual with ease.

An obvious solution is to adopt stronger authentication methods. Unfortunately, businesses have little incentive to do so. Defrauded sellers typically roll their losses due to identity crime into the cost of doing business, in effect recovering these costs from honest customers. Thus, the added costs of strong authentication methods would not be offset either by a reduction in losses or

an increase in profits. We might hope that stronger authentication methods would be attractive to customers, thereby creating a competitive advantage for any institution that deploys such authentication methods, since dealing with the loss of reputation and tarnished credit ratings is both time consuming and frustrating for victims of identity crimes. However, historically customers have been unwilling to trade convenience or incur higher costs for security.

Governments usually step in when incentives are needed to compel behaviors required for the greater good of our society. Legislation to require strong authentication for certain transactions would go a long way to eliminating identity crime but would be unpopular with business (because of the costs) and with individuals (because of the inconvenience). So instead consumers are admonished to keep secret their social security numbers, credit-card numbers, and bank account numbers. And legislation is being passed that incentivizes business to better protect files containing such identifiers; data breaches are then less likely to give criminals access to this information.[37] These measures can be seen as an attempt to change the culture, creating an environment where criminals no longer have easy access to the identifiers they require to spoof today's (weak) authentication mechanisms. But identifiers are, by their nature, not secrets; therefore, secrecy of identifiers can be a temporary stop gap, at best. Widespread acceptance and deployment of strong forms of authentication is what's needed.

5.5 Federation and Single Sign-On

Whatever privacy benefits accrue from having multiple identities, managing all those identities can be burdensome. You must keep track of what attributes are included in each identity, and you must keep track of what identity you are using for each *service provider* (also known as a *relying party*) with which you interact. In particular, a new identity must be created or an existing identity selected whenever a new service provider is contacted, where that identity complies with the Select Minimal Identity and Avoid Linking guidelines of §5.4.1.

Some of this burden can be eliminated by using an *identity provider* to manage identities and store their attributes. The identity provider would authenticate a user once and thereafter serve as that user's proxy, choosing a suitable identity for each of that user's interactions with various service providers. Continuity of a relationship is supported by choosing the same identity for related interactions; linking is avoided by choosing a different identity or creating a new identity for an interaction.

Service providers that were designed to authenticate a human user directly must typically undergo *provisioning* before they are able to operate in an environment where interposed identity providers speak for human users:

[37] However, numerous data sources beyond stolen laptops and electronic databases help criminals commit identity theft, and we have little understanding of where criminals are getting the credentials they use.

- A trust relationship between the service provider and the identity provider might have to be established if one does not already exist. This typically involves legal negotiations, culminating in a contract to spell out responsibilities (e.g., about providing notice and safeguarding for personal information) and define liability limits for the service provider and identity provider.

- Service provider programming might have to be modified. A web server, for example, in responding to a user's request might employ a temporary redirect command[38] in order to force the user's web browser to invoke an identity provider and obtain credentials, which are then included in a new, augmented request to the web server.

- An exchange of cryptographic keys might be necessary, so the identity provider and service provider can authenticate each other.

The simplicity of having a single, system-wide identity provider is attractive but is impractical. Business, personal, and/or political alliances inevitably bias users and service providers against all sharing the same identity provider. Also, a user's different identities might warrant different levels of protection, in which case no single identity provider would be well suited to handle all.

Federation can hide much of the complexity that having multiple identity providers brings. The goal of federating is to give the appearance of a single, monolithic, identity provider. Federation administrators define (i) a single set of meanings for attributes used in identities, (ii) accuracy standards for what attributes store, (iii) acceptable ways for storing and exchanging sensitive information, (iv) characteristics required (or prohibited) of federation members, and (v) a single legal instrument governing accountability and obligations between federation members and with service providers. Agreement on all these points then makes it possible for identity providers in a federation to implement a common suite of protocols, including protocols a user (or user-agent software) employs to authenticate with identity providers and protocols an identity provider employs for a user (or user-agent software) to authenticate under a given identity to a specific service provider.

Metaphors for Identity

Without some sort of higher-level metaphor as the basis for the identity provider's user interface, human users are unlikely to reap the privacy benefits that having multiple identities affords. We sketch two such metaphors below—one based on access control, and a second based on identity cards. Both are intended to foster usability by making it easier for users to anticipate the consequences of their actions.

[38] HTTP command 302.

Access Control Metaphor. Users concerned with privacy are apt to feel particularly comfortable with a metaphor that focuses on their specifying

- which attributes are allowed and not allowed to be seen by each specific service provider, and

- under what circumstances and how may attributes be used, including whether and where they may be stored, for how long, and for what purposes.

This metaphor views identity management as an access control problem.

A user, by specifying a policy about whether a given service provider can be sent each of that user's attributes, implicitly constrains which identities may be used for interacting with that service provider. And by specifying what requests are part of a given long-term relationship with some set of service providers versus requests that should be seen as uncorrelated, a user defines when an existing identity can be reused or a new one must be created. Other constraints could be added automatically by user-agent software, to incorporate knowledge about attributes involving personally identifiable information, hence governed by privacy legislation, and knowledge about information that was expected to remain secret but nevertheless could be inferred from information the user's policies allow to be revealed. Notice, an identity provider can extract constraints on identity usage automatically from these policies about attributes and requests, freeing users from having to select identities for interactions with service providers.

For example, a user U might interact with an on-line retailer *OLR*, a shipper *SHP*, and the bank *BNK* that issued U's credit card. If U asserts that all book purchases at *OLR* be seen as correlated, then this should cause U's identity provider to employ the same identity in accessing *OLR* for all book purchases but to use a different identity for U's purchases of other kinds of articles. U might also stipulate that shipper *SHP* learn the delivery address for a purchase but not the contents of any package it delivers for U, that *OLR* not learn the delivery address for each purchase, and that *BNK* not know the particulars of items being charged to the credit card *BNK* issues. Given these constraints, U's identity provider would deduce that three separate identities must be used for interacting with *OLR*, *SHP*, and *BNK*, because different sets of attributes must be omitted in each.

There is considerable flexibility, though, in how *OLR*, *SHP*, and *BNK* communicate for this purchase transaction. One obvious (but high-latency and unnecessarily centralized) implementation has *OLR* forward a separate request to U's identity provider when *SHP* and *BNK* each need to be engaged; the identity provider then interacts directly with *SHP* and *BNK*. A more attractive implementation has U's identity provider encrypting the three attribute bundles for the three identities being used, and doing so in a way that allows each bundle to be decrypted only by the corresponding service provider. The three bundles are then forwarded to *OLR* and conveyed to/by *SHP* and/or *BNK* as the purchase transaction proceeds.

Identity Card Metaphor. A second metaphor, which should appeal to technologically unsophisticated users, is based on the various *identity cards* that can be found in a person's wallet: driver's license, university id, various credit cards, etc. In the physical world, each of these identity cards represents a different identity, and by selecting among the identity cards, different identities can be utilized for different transactions. We might therefore portray identity management in familiar terms simply by positing a *virtual identity card* for each identity, associating a set of attributes with each virtual identity card, and allowing users to store and manipulate these virtual identity cards. As before, users would somehow specify:

- which identity cards may be seen and must not be seen by each specific service provider, and
- under what circumstances and how may an identity card be used, including whether and where they may be stored, for how long, and for what purposes.

What makes virtual identity cards an attractive metaphor is that most acts involving real identity cards have natural parallels as actions involving virtual identity cards. Consider a user interface that represents each virtual identity card by using an icon depicting an identity card. Then dragging that icon onto a graphical representation of some service provider S will be seen by many as an intuitive way to indicate an identity for interacting with S. Moreover, the metaphor leverages everyday experience that linking is possible when the same identity card is used repeatedly and that all identity cards are not equally appropriate for all uses—both of which are truths about virtual identity cards and user identities, as well.

Exercises for Chapter 5

5.1 In Western culture, a hyphenated surname often indicates the surnames of a person's parents. Other conventions exist in other cultures. Describe the schemes used in Iceland and in India.

5.2 Given a set comprising N characters, there are N^L sequences of length L. Give a formula for the number of sequences having length less than or equal to L.

5.3 You have been asked to determine the feasibility of conducting a brute-force attack on a well-chosen password. How long would it take a 2 GHz processor to enumerate all strings comprising 10 or fewer characters, assuming a character set of 52 lower and upper-case characters plus the 10 digits found on a computer keyboard?

5.4 Linguists claim that English has 1.3 bits of information per character.

(a) Explain the reason for the discrepancy between this claim and a naive information-theoretic analysis, which would assert that English has $\log_2 26$ bits of information per character.

(b) Suppose passwords constructed entirely from English words are desired. If a password is intended to be as strong as a 64 bit random string, how many total characters must the password be?

5.5 An alternative to storing sets of pairs $\langle uid_i, \mathcal{H}(pass_i)\rangle$ is to protect the confidentiality of passwords by storing a set of hashed pairs $\mathcal{H}(\langle uid_i, pass_i\rangle)$. Give the process for checking whether a correct password *pass* has been entered for a given user identifier *uid*. What are the advantages and/or disadvantages of this alternative?

5.6 In Microsoft's Windows NT operating system, two password hashes are used. One is called the LM or LAN Manager hash, and it is present for backward compatibility with Windows 95/98. Here's how the LM hash is computed for a 14 character input:

(i) All characters in the input are transformed into upper case, and the result $C = c_1c_2 \ldots c_{14}$ is split into two 7 character halves $C' = c_1c_2 \ldots c_7$ and $C'' = c_8c_9 \ldots c_{14}$.

(ii) C' and C'' are separately "hashed" by converting each into a DES key and encrypting the ASCII string `KGS!@#$%` to produce two 8 byte values H' and H''.

(iii) Concatenating $H' \cdot H''$ yields the 16 byte hash H of the original input.

Compare the work required for a brute-force attack against a 14 character password whose LM hash is being stored with the work required if a single hash of the entire 14 character input were used.

5.7 Consider the following alternatives to storing set *HashSsndPwd* of tuples $\langle uid_i, salt_i, \mathcal{H}(pass_i \cdot salt_i \cdot ppr_i)\rangle$, where $salt_i$ is salt and ppr_i is pepper. For each alternative:

(i) Exhibit the protocol for checking whether a password is valid for a given user identifier.

(ii) Compare the alternative to the use of *HashSsndPwd*, discussing whether (and how) it is better or worse.

(a) Store a set of pairs $\langle uid_i, \mathcal{E}^{DES}_{pass_i}(uid_i \cdot pad(uid_i))\rangle$ where $\mathcal{E}^{DES}_{pass_i}(\cdot)$ is the DES encryption function with a key derived from $pass_i$, and $pad(uid_i)$ is a well known function to generate padding given the length of uid_i.

(b) Store a set of triples $\langle uid_i, \mathcal{E}^{DES}_{pass_i}(n_i), \mathcal{H}(pass_i \cdot n_i)\rangle$ where $\mathcal{E}^{DES}_{pass_i}(\cdot)$ is the DES encryption function with a key derived from $pass_i$, and $\mathcal{H}(\cdot)$ is a hash function.

5.8 Describe an on-line guessing attack that is possible when passwords are stored in a set *HashPwd* of pairs $\langle uid_i, \mathcal{H}(pass_i)\rangle$ but that no longer works if passwords are stored in a set *HashSaltPwd* of triples $\langle uid_i, n_i, \mathcal{H}(pass_i \cdot n_i)\rangle$.

5.9 One-time passwords are an effective defense against shoulder surfing and other forms of passive wire-tapping. But people are not good at memorizing long lists, so other means must be employed to generate the one-time passwords. Here is a somewhat unconventional proposal, based on a scheme suggested by Weinshall [35].

Choose security parameters m and n, where $m < n$ holds. Each user memorizes a secret, which is a set of m indices, and shares this secret with the authenticating computer. The indices identify m Boolean variables from a larger set of n Boolean variables.

To be authenticated, the user is shown a random collection of k Boolean formulas over the n Boolean variables. This collection of formulas is constructed so that at least one formula evaluates to *true* when all of the user's m variables equal *true* and the other $n - m$ variables equal *false*.

The user responds by entering the value (*true* or *false*) for each of the formulas. The system authenticates the user if and only if that user's responses demonstrate knowledge of the m secret indices.

(a) How many guesses must an attacker make, on average, to be authenticated after watching the user successfully be authenticated r times?

(b) How does the answer to (a) change if the every collection of formulas presented to the user has exactly one formula that equals *true* (as opposed to at least one formula)?

5.10 The protocol of Figure 5.7 requires authentication token $\mathcal{A}_P$ to store PIN PIN_P. Modify the protocol so this PIN per se is not be stored anywhere, but nevertheless a PIN must be entered into $\mathcal{A}_P$ in order for an authentication to succeed.

5.11 With the protocol of Figure 5.7, an attacker that compromises authenticating computer *Sys* learns all of the secrets in *tokenSec*, which would then allow that attacker to impersonate authentication token $\mathcal{A}_P$ for any user P. Modify the protocol of Figure 5.7 and use public key cryptography in a way that defends against such attacks.

5.12 In the protocol of Figure 5.7, authenticating computer *Sys* stores its challenge r for later reference in step 5. An alternative to storing r locally has *Sys* encrypt r and include this ciphertext along with r in the original challenge of step 1; $\mathcal{A}_P$ would then return this ciphertext as part of its response in step 4. The result is a stateless protocol, which is preferable. What, if any, vulnerabilities does the protocol modification create?

5.13 The time-based authentication token protocol of Figure 5.8 involves a set *Psbl*, which is defined in terms of the maximum message transmission delay L

that could be measured by any clock. Derive an alternative definition for *Psbl* that instead uses a real-time bound l on delivery delay.

5.14 Using the definitions $m_N = \mathcal{H}(s_P)$, and $m_i = \mathcal{H}(m_{i+1})$ for $1 \leq i < N$, give a proof of (5.13):

$$\text{For } 1 \leq i \leq N : m_i = \mathcal{H}^{N-i+1}(s_P)$$

5.15 Can test (5.16) be replaced by the following?

$$i_P > i^P_{Sys} \ \wedge \ \mathcal{H}^{i^P_{Sys} - i_P}(last_P) = resp \tag{5.17}$$

If so, give a proof that (5.17) implies (5.16); if not explain why the alternative test cannot be used.

5.16 Instead of using hash chain (5.12), we might instead define $m_i = \mathcal{H}(i \cdot s_p)$. Discuss the merits and vulnerabilities of this alternative relative to using the hash chain. In short, what (if anything) does the hash chain bring to solving this authentication problem?

5.17 Suppose a poor hash function were used in the protocol of Figure 5.10 and, therefore, a value is likely to recur in the sequence $\mathcal{H}^1(s_p)$, $\mathcal{H}^2(s_p)$, ..., $\mathcal{H}^{1000}(s_p)$. Does this lead to a vulnerability? If so, describe an attack to exploit that vulnerability; if not, explain why no new vulnerability is introduced.

5.18 What is the effect of replacing step 4 of Figure 5.10 with the following?

4. $\mathcal{A}_P \longrightarrow Sys$: $\langle P, r+10, resp \rangle$ where $resp = \mathcal{H}^{N-(i_P+10)+1}(s_P)$

Are any vulnerabilities introduced? Will the protocol continue to function as before?

5.19 The *true match rate* (TMR) for a biometric sensor is defined to be $1-FRR$, where FRR is the false reject rate; and the *true non-match rate* (TNMR) is defined to be $1 - FAR$, where FAR is the false accept rate.

An individual is granted access whenever there is a match with somebody whose biometric is stored. If the TMR and TNMR for a biometric sensor both are 99.9% then how often will an impostor be granted access?

5.20 Consider the following scheme for preventing attackers who succeed in reading the biometric templates stored on *Sys* from learning enough to create a bogus sensor value val_U for impersonating a user U.

> **Hash Template Storage.** *Sys* stores a hash $\mathcal{H}(t_U)$ instead of the template t_U for each user U. And rather than checking whether $match(t_U, v_U)$ holds for measured sensor value v_U, *Sys* checks $match(\mathcal{H}(t_U), \mathcal{H}(v_U))$. □

Is this scheme feasilble? Explain why.

5.21 To what extent is the Select Minimal Identity guideline given on page 73 a corollary of the Principle of Least Privilege given on page **??**.

5.22 Inspired by the new U.S. passports, Cornell has decided to upgrade its id cards to use RFID chips instead of magnetic stripes. (Magnetic stripes wear out; passive RFID chips don't.) Magnetic-card readers outside building doors will be replaced by radio transceivers that can interrogate the new RFID-enabled id cards up to a distance of ten feet, even while the card remains safely ensconced in a pocket. The new scheme is clearly more convenient. Discuss the extent to which the new scheme complies with the guidelines in §5.4.1 for deploying authentication.

Notes and Reading

The idea that methods for authenticating people could be divided into three basic approaches—something you know, something you have, or something you are—is discussed in a 1970 IBM technical report [11]. But only recently have computer scientists turned their attention to studying identity, its characterization as a set of attributes, and its connection with authentication. Drivers for this line of inquiry include the growing sensitivity about both privacy and the inconveniences of interacting with ever larger numbers of distinct systems. Much has been written on the subject; our treatment of the topic is based on the 2003 National Academy report *Who Goes There? Authentication Through the Lens of Privacy* [21].

The term CAPTCHA first appeared in 2000 [32], and CAPTCHA's are widely used by web sites to defend against bots and other programs misappropriating services that were intended for use only by humans. However, the idea of posing certain kinds of challenges as a way to distinguish people from computers dates back to a 1996 technical report by Naor [28], where various visual-recognition tasks (gender recognition, facial expression understanding, naming body parts, deciding nudity) illustrate the approach.

Passwords have been the canonical approach for authenticating people since the advent of time-sharing. MIT's Compatible Time Sharing System (CTSS) [10] by summer 1964 had passwords to authenticate users.[39] Wilkes, in connection with his work on the Cambridge Titan system [38], was the first to discuss storing passwords in encrypted form (crediting Needham for the idea) Salt was introduced in UNIX as a defense against off-line dictionary attacks [27]. And pepper was independently proposed by Manber [26] and by Abadi et al. [2]; it is not widely used.

A good deal has been written about good choices of passwords. The rules of Figure 5.1 are based on rules given in Bishop [5], which were derived from a 1990 study conducted by Klein [22], who analyzed password files comprising some 15,000 entries.

Other aspects of password security came to be recognized only as password-based authentication was deployed. Password expiration, for example, was not implemented by CTSS and was only added to Multics (MIT's successor to CTSS) as part of Project Guardian, a USAF-funded effort from 1972 through

[39] CTSS first became operational, however, three years earlier in 1961.

1974 to make Multics more secure. And even in the mid-1970's, Multics did not have a trusted path[40] although users accessing the system over dial-up telephone lines were advised to disconnect and re-dial at the start of each session to ensure their terminal was not communicating with a password harvester that had been left running. With time, however, the wisdom of a system-supported trusted path became clear, and by August 1983 the United States DoD standard for evaluating computer security [8]—known as the "Orange Book" from the color of its cover—was unequivocal about requiring operating system support for a trusted path.

Attacks to steal passwords (and other information) that humans enter using a keyboard and/or are displayed on monitors have coevolved with new technology (which open new vulnerabilities) and defenses (which close old vulnerabilities). Low-cost computers with the power to execute signal processing algorithms made it feasible to reconstruct what a user types by sensing and analyzing keyboard acoustic emanations [4, 40] and what is being displayed from reflections of a CRT screen on the walls of a room [23] or from electromagnetic signals that flat-panel displays radiate [24]. Remote access to computers over public networks meant passwords sent in the clear could be intercepted in transit, which in turn led to defenses based on one-time passwords. Lamport's 1979 hash chain scheme [25] was the first; it is the basis for the S/KEY software developed at Bellcore in the late 1980's by Haller, Karn, and Walden [18, 17] and the Digital Pathways SNK calculator. The use of synchronized clocks for generating one-time passwords is described in a series of patents [36, 37].

Willie Sutton is frequently misquoted as saying that he robbed banks "because that's where the money is," and the growth of phishing attacks instantiates that observation for online banking and commerce using the web. One of the more influential early papers about phishing was written by Felten et al. [13]; it credits Cohen [9] with the basic attack technique. Password hashes were developed independently by Abadi et al. [1] and Gabber et al. [15] to block phishing, since a password sent to one web site would then be useless for authenticating at other sites. See the description of `PwdHash` [29] for solutions to the engineering and user-interface issues associated with integrating password hashes into a modern web browser. A use of pictures to prevent phishing is described by Dhamija and Tygar [12]. Here, a picture is provided by a web site that requests a password so the site can be authenticated by whomever must supply the password, but subsequent experiments [20] suggest humans are easily fooled when authentication of a web site (or determining that a secure communications protocol like `https` is in use) requires recognizing some visual secret.

Of the various technologies suggested for use in authentication tokens, RFID tags have attracted the most attention. The idea of transmitting information by modulating radio signal reflections was first employed by Luftwaffe pilots during World War II, the first war where radar was in use. German radar operators found that the radar reflections from an airplane being rolled in flight were distinguishable from what would be displayed for a plane being flown nor-

[40] The Multics `ATTN` key was delivered to a user process, not the system's login agent.

mally. So Luftwaffe pilots returning to base in Germany would roll their planes to signal that they were friendly. Stockman [31] published the first mathematical account of this phenomon in 1948, validating his theory with experimental results for various modulation methods. But not until the 1970's were passive transponders for radio signals developed and fielded—for controlling access to doors, for tracking trucks carrying nuclear material, and ultimately as the basis for automatic highway toll collection (now quite common in U.S. cities). The volume [16] of short papers representing various stakeholders' opinions is a good starting place for those seeking a more in depth treatment of modern RFID technology and its implications.

The literature on biometrics is enormous, with interest in the subject predating computers by centuries. Fingerprinting, for example, traces its roots to 9th century China, where it was used to authenticate merchants participating in transactions; the first use of fingerprints for identifying criminals occurred in India, late in the 19th century. The tutorial on biometrics in §5.3 is based on two books [39, 33], but the survey by Wayman [34] is also a good resource.

With scientific papers about federated identity and single sign-on relatively scarce, descriptions of extant systems is a good way to learn about the design space. Cameron's Identity Weblog [6] is also a useful resource. Kerberos [30], designed and fielded long before the web, is still widely used for providing single sign-on to services in an enterprise network. Microsoft Passport, a single sign-on system for users to access many web sites without having to login to each, is generally regarded as a failure (and worth studying for that reason). Notable successors to these earlier efforts include: Cardspace [7], Liberty Alliance [3], OpenID [14], and Shibbolith [19].

Bibliography

[1] Martin Abadi, Krishna Bharat, and Johannes Marais. System and method for generating unique passwords. U.S. Patent 6,141,760. Filed October 1997, issued October 2000.

[2] Martín Abadi, T. Mark A. Lomas, and Roger Needham. Strengthening passwords. Technical Report 1997–033, Digital Equipment Corporation, 1997.

[3] Liberty Alliance. `http://www.projectliberty.org/`.

[4] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–11, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[5] Matt Bishop. *Computer Security: Art and Science*. Pearson Education Inc., Boston, MA, 2002.

[6] Kim Cameron. Identity weblog. `http://www.identityblog.com/home.php/`.

[7] Kim Cameron and Michael B. Jones. Design rationale behind the identity metasystem architecture. In Norbert Pohlmann, Helmut Reimer, and Wolfgang Schneider, editors, *ISSE/SECURE 2007 Securing Electronic Business Processes Highlights of the Information Security Solutions Europe/SECURE 2007 Conference*, pages 117–129. Vieweg, 2007.

[8] National Computer Security Center. Trusted computer system evaluation criteria. Technical Report CSC-STD-001-83, Department of Defense, August 1983.

[9] Fred Cohen. 50 ways to attack your world wide web system. In *Computer Security Industry Annual Conference*, October 1995.

[10] Fernando J. Corbató, Marjorie Merwin-Daggett, and Robert C. Daley. *An experimental time-sharing system*, pages 117–137. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[11] IBM Corporation. The consideration of data security in a computer environment. Technical Report G520-2169, IBM Corporation, 1970.

[12] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security*, pages 77–88, New York, NY, USA, 2005. ACM Press.

[13] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web spoofing: An internet con game. In *Proceedings of 20th National Information Systems Security Conference*, October 1997.

[14] OpenID Foundation. `http://openid.net/`.

[15] Eran Gabber, Phillip B. Gibbons, Yossi Matias, and Alain J. Mayer. How to make personalized web browsing simple, secure, and anonymous. In Rafael Hirschfeld, editor, *Proceedings of First International Conference Financial Cryptography FC'97*, volume 1318/1997 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 1997.

[16] Simson Garfinkel and Beth Rosenberg, editors. *RFID Applications, Security, and Privacy*. Addison-Wesley, 2005.

[17] N. Haller. The S/KEY One-Time Password System. RFC 1760 (Informational), February 1995.

[18] Neil M. Haller. The S/KEY one-time password system. In *Proceedings of the ISOC Symposium on Network and Distributed System Security*, pages 151–157, 1994.

[19] Internet2 Middleware Initiative. `http://shibboleth.internet2.edu/`.

[20] Collin Jackson, Daniel R. Simon, Desney S. Tan, and Adam Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 281–293. Springer, 2007.

[21] Stephen T. Kent and Lynette I. Millet, editors. *Who Goes There? Authentication Through the Lens of Privacy*. National Academies Press, 2003.

[22] Daniel V. Klein. Foiling the cracker—A survey of, and improvements to, password security. In *Proceedings of the Second USENIX Workshop on Security*, pages 5–14, Summer 1990.

[23] Markus G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–18, Washington, DC, USA, 2002. IEEE Computer Society.

[24] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In David Martin and Andrei Serjantov, editors, *Proceedings 4th Workshop on Privacy Enhancing Technologies*, volume 3424 of *Lecture Notes in Computer Science*, pages 23–25. Springer, May 2004.

[25] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[26] Udi Manber. A simple scheme to make passwords based on one-way functions much harder to crack. *Computers and Security*, 15(2):171–176, 1996.

[27] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[28] Moni Naor. Verification of a human in the loop or identification via the turing test, 1996.

[29] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *SSYM'05: Proceedings of the 14th Conference on USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2005. USENIX Association.

[30] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202, Berkeley, CA, 1988. USENIX Association.

[31] Harry Stockman. Communication by means of reflected power. *Proceedings of the IRE*, 36(10):1196–1204, October 1948.

[32] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langfort. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311. Springer-Verlag, 2003.

[33] James Wayman, Anil Jain, Davide Maltoni, and Dario Maio, editors. *Biometric Systems: Technology, Design, and Performance*. Springer, 2005.

[34] James L. Wayman. Biometrics in identity management systems. *IEEE Security and Privacy*, 6(2):30–37, March/April 2008.

[35] Daphna Weinshall. Cognitive authentication schemes safe against spyware (short paper). In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 295–300, Washington, DC, USA, 2006. IEEE Computer Society.

[36] Kenneth P. Weiss. Method and apparatus for positively identifying an individual. U.S. Patent 4,720,860. Filed November 1984, issued January 1988.

[37] Kenneth P. Weiss. Method and apparatus for synchronizing generation of separate, free running, time dependent equipment. U.S. Patent 4,885,778. Filed November 1985, issued December 1989.

[38] M. V. Wilkes. *Time Sharing Computer Systems*. Elsevier Science Inc., New York, NY, USA, 1968.

[39] John D. Woodward, Jr., Nicholas M. Orlans, and Peter T. Higgens. *Biometrics*. McGraw-Hill/Osborne, 2003.

[40] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In *CCS '05: Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 373–382, New York, NY, USA, 2005. ACM Press.